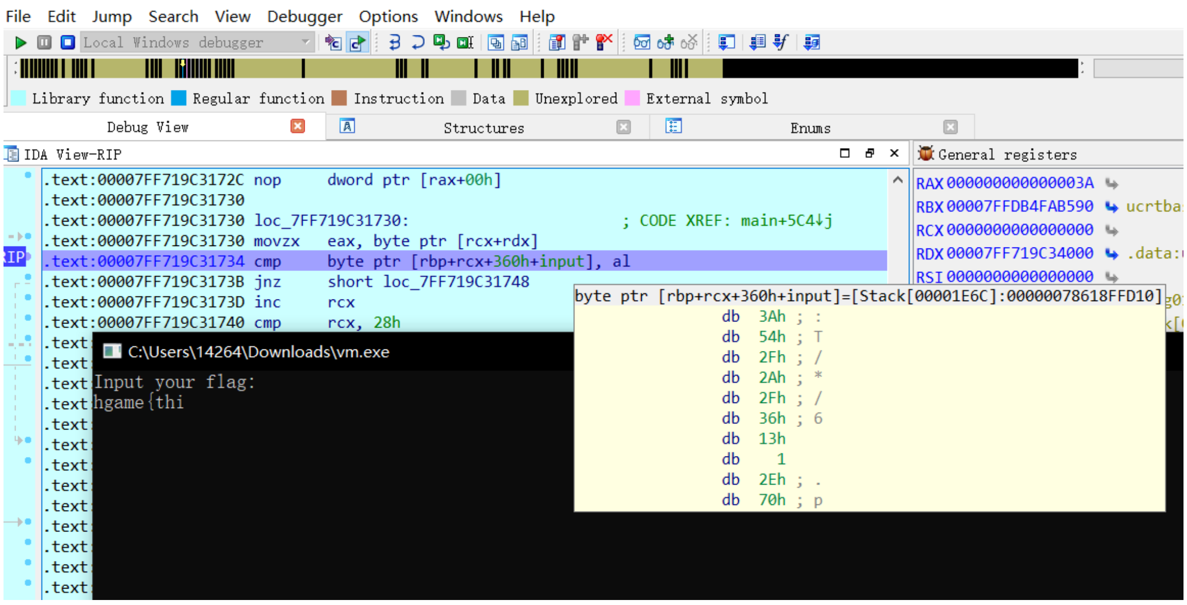
RE

easyVM

调试的时候发现是将输入经过变换后和一串字符进行比较，并且后面的输入不会影响前面的结果，于是就。。。。

根据flag的格式套路把flag凑出来了



```
.text:00007FF719C3172C nop     dword ptr [rax+00h]
.text:00007FF719C31730
.text:00007FF719C31730 loc_7FF719C31730:                       ; CODE XREF: main+5C4↓j
.text:00007FF719C31730 movzx   eax, byte ptr [rcx+rdx]
.text:00007FF719C31734 cmp     byte ptr [rbp+rcx+360h+input], al
.text:00007FF719C3173B jnz     short loc_7FF719C31748
.text:00007FF719C3173D inc     rcx
.text:00007FF719C31740 cmp     rcx, 28h
```

```
RAX 000000000000003A
RBX 00007FFDB4FAB590  ucrtba
RCX 0000000000000000
RDX 00007FF719C34000  .data:
RSI 0000000000000000
```

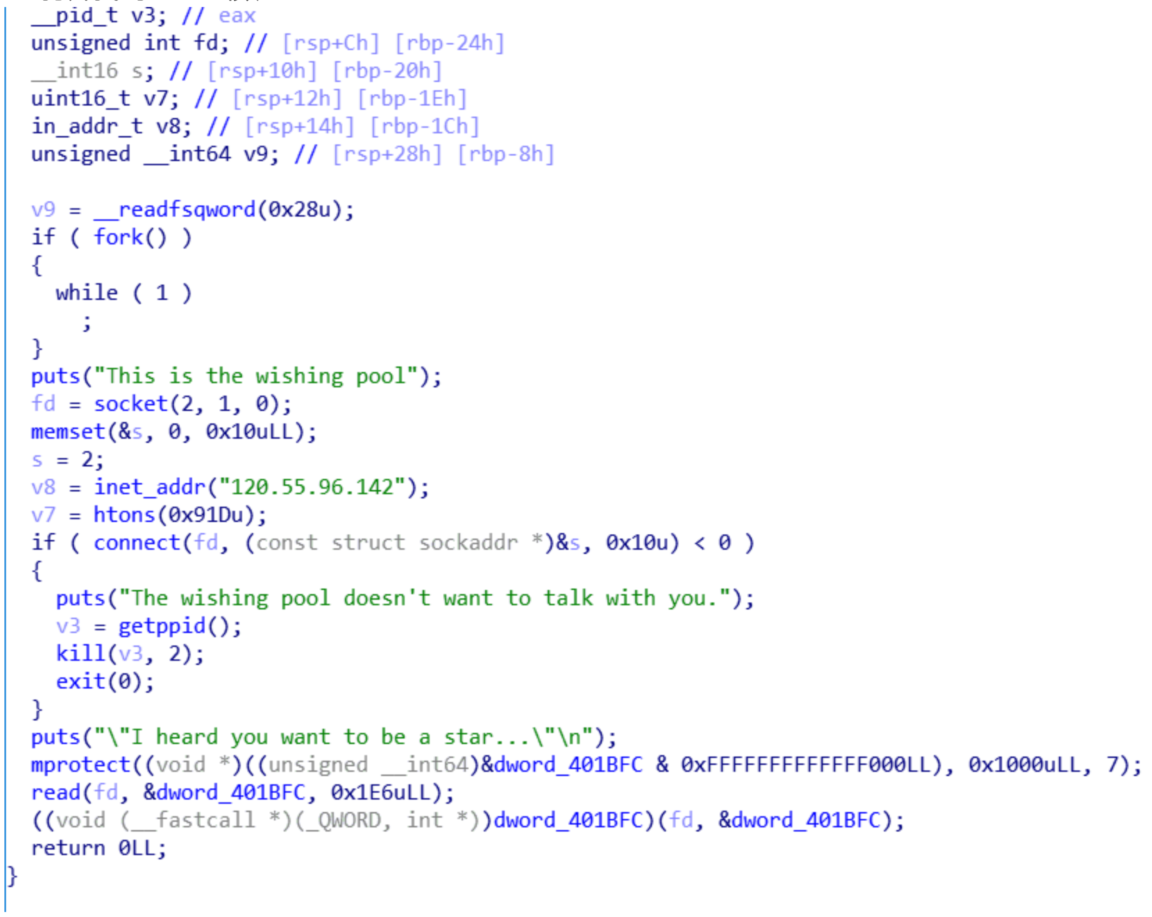
```
Input your flag:
hgame{thi
```

```
byte ptr [rbp+rcx+360h+input]=[Stack[00001E6C]:00000078618FFD10]
                db  3Ah ; :
                db  54h ; T
                db  2Fh ; /
                db  2Ah ; *
                db  2Fh ; /
                db  36h ; 6
                db  13h
                db    1
                db  2Eh ; .
                db  70h ; p
```

出题人对不起！！！！

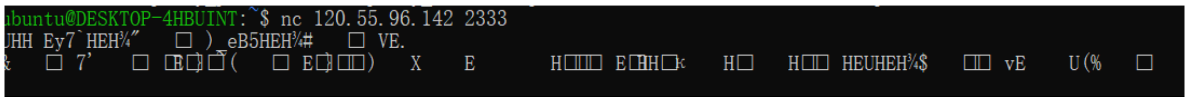
##Secret

ida打开找到main函数

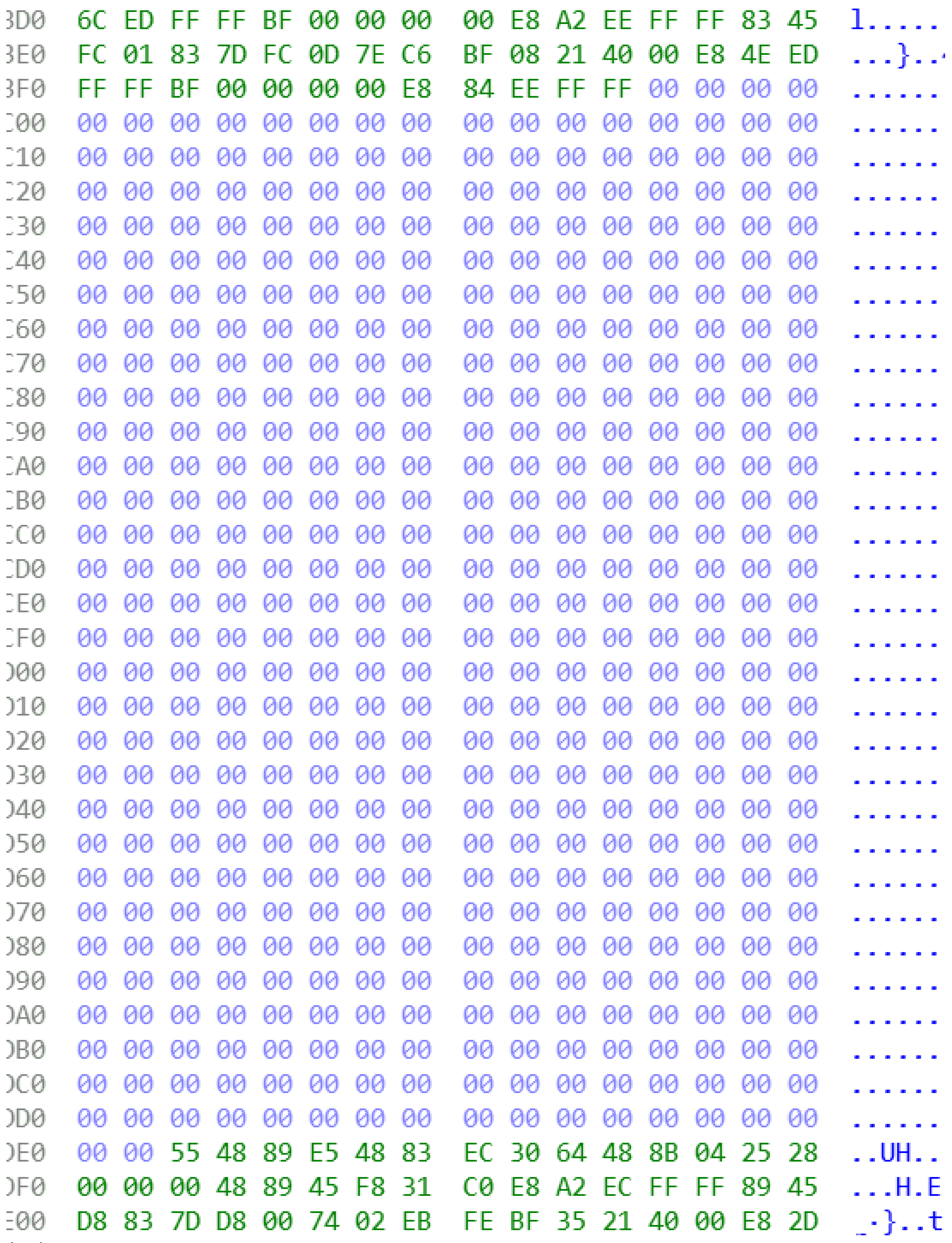
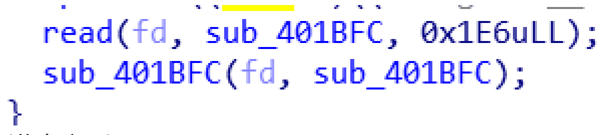


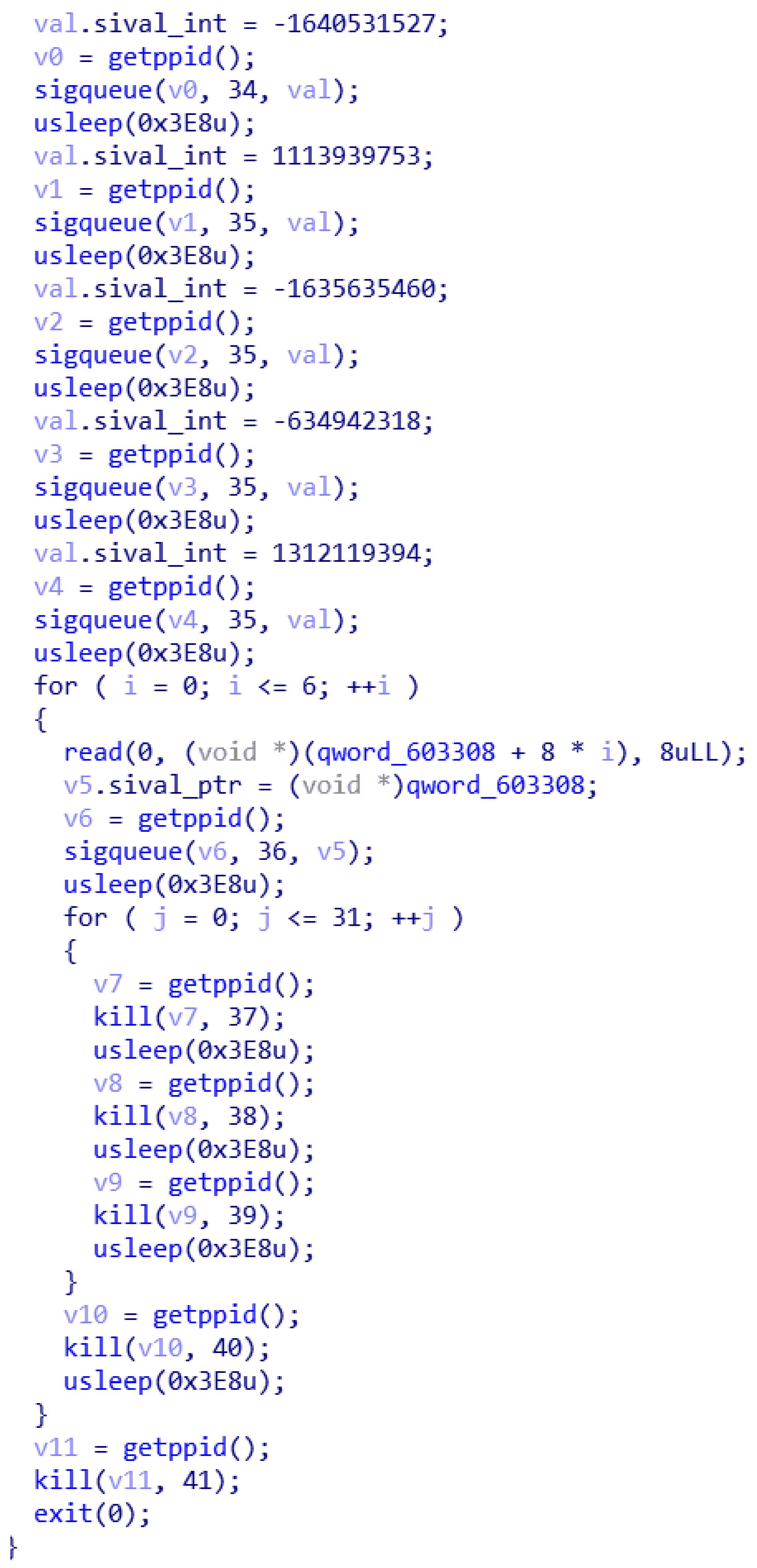
```
  __pid_t v3; // eax
  unsigned int fd; // [rsp+Ch] [rbp-24h]
  __int16 s; // [rsp+10h] [rbp-20h]
  uint16_t v7; // [rsp+12h] [rbp-1Eh]
  in_addr_t v8; // [rsp+14h] [rbp-1Ch]
  unsigned __int64 v9; // [rsp+28h] [rbp-8h]

  v9 = __readfsqword(0x28u);
  if ( fork() )
  {
    while ( 1 )
      ;
  }
  puts("This is the wishing pool");
  fd = socket(2, 1, 0);
  memset(&s, 0, 0x10uLL);
  s = 2;
  v8 = inet_addr("120.55.96.142");
  v7 = htons(0x91Du);
  if ( connect(fd, (const struct sockaddr *)&s, 0x10u) < 0 )
  {
    puts("The wishing pool doesn't want to talk with you.");
    v3 = getppid();
    kill(v3, 2);
    exit(0);
  }
  puts("\"I heard you want to be a star...\"\n");
  mprotect((void *)((unsigned __int64)&dword_401BFC & 0xFFFFFFFFFFFFF000LL), 0x1000uLL, 7);
  read(fd, &dword_401BFC, 0x1E6uLL);
  ((void (__fastcall *)(_QWORD, int *))dword_401BFC)(fd, &dword_401BFC);
  return 0LL;
}
```

发现使用了soket，用nc连接



```
ubuntu@DESKTOP-4HBUINT:~$ nc 120.55.96.142 2333
UHH Ey7`HEH¾"    □ )_eB5HEH¾#    □ VE.
&    □ 7'    □  □B□□□(    □ E□□□)    X    E        H□□□ E□HH□k    H□    H□□ HEUHEH¾$    □□ vE    U(%    □
```

是一串乱码
用python算了一下这串乱码的长度，是0x1E6，正好和read函数读取的长度一样
然后看了下这个文件的hex，在401bfc的位置发现了一串0

```
3D0  6C ED FF FF BF 00 00 00  00 E8 A2 EE FF FF 83 45  l.....
3E0  FC 01 83 7D FC 0D 7E C6  BF 08 21 40 00 E8 4E ED  ...}..
3F0  FF FF BF 00 00 00 00 E8  84 EE FF FF 00 00 00 00  ......
C00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C80  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
C90  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
CA0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
CB0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
CC0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
CD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
CE0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
CF0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D00  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D10  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D20  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D30  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D40  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D50  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D60  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D70  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D80  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
D90  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
DA0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
DB0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
DC0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
DD0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ......
DE0  00 00 55 48 89 E5 48 83  EC 30 64 48 8B 04 25 28  ..UH..
DF0  00 00 00 48 89 45 F8 31  C0 E8 A2 EC FF FF 89 45  ...H.E
E00  D8 83 7D D8 00 74 02 EB  FE BF 35 21 40 00 E8 2D  ..}..t
```
长度是0x1E6
程序原本是不完整的，要通过soket读取一串字符后把它写到相应的地方，再把这块内存修改为7
填上这块0之后发现最后一句变成了这样

```
  read(fd, sub_401BFC, 0x1E6uLL);
  sub_401BFC(fd, sub_401BFC);
}
```
进去之后



```
  val.sival_int = -1640531527;
  v0 = getppid();
  sigqueue(v0, 34, val);
  usleep(0x3E8u);
  val.sival_int = 1113939753;
  v1 = getppid();
  sigqueue(v1, 35, val);
  usleep(0x3E8u);
  val.sival_int = -1635635460;
  v2 = getppid();
  sigqueue(v2, 35, val);
  usleep(0x3E8u);
  val.sival_int = -634942318;
  v3 = getppid();
  sigqueue(v3, 35, val);
  usleep(0x3E8u);
  val.sival_int = 1312119394;
  v4 = getppid();
  sigqueue(v4, 35, val);
  usleep(0x3E8u);
  for ( i = 0; i <= 6; ++i )
  {
    read(0, (void *)(qword_603308 + 8 * i), 8uLL);
    v5.sival_ptr = (void *)qword_603308;
    v6 = getppid();
    sigqueue(v6, 36, v5);
    usleep(0x3E8u);
    for ( j = 0; j <= 31; ++j )
    {
      v7 = getppid();
      kill(v7, 37);
      usleep(0x3E8u);
      v8 = getppid();
      kill(v8, 38);
      usleep(0x3E8u);
      v9 = getppid();
      kill(v9, 39);
      usleep(0x3E8u);
    }
    v10 = getppid();
    kill(v10, 40);
    usleep(0x3E8u);
  }
  v11 = getppid();
  kill(v11, 41);
  exit(0);
}
```

看到了一串不认识的函数

经百度后发现这是用来发送信号的，又再函数列表里找到了接受信号的函数

```
v5 = __readfsqword(0x28u);
sigemptyset((sigset_t *)&v2);
v1 = sub_4019EA;
v3 = 4;
sigaction(34, (const struct sigaction *)&v1, (struct sigaction *)&v4);
sigemptyset((sigset_t *)&v2);
v1 = sub_401A09;
v3 = 4;
sigaction(35, (const struct sigaction *)&v1, (struct sigaction *)&v4);
sigemptyset((sigset_t *)&v2);
v1 = sub_401A3A;
v3 = 4;
sigaction(36, (const struct sigaction *)&v1, (struct sigaction *)&v4);
v1 = sub_401AA4;
v3 = 0;
sigaction(37, (const struct sigaction *)&v1, (struct sigaction *)&v4);
v1 = sub_401AF6;
v3 = 0;
sigaction(38, (const struct sigaction *)&v1, (struct sigaction *)&v4);
v1 = sub_401B11;
v3 = 0;
sigaction(39, (const struct sigaction *)&v1, (struct sigaction *)&v4);
v1 = sub_401B66;
v3 = 0;
sigaction(40, (const struct sigaction *)&v1, (struct sigaction *)&v4);
v1 = sub_401B9D;
v3 = 0;
sigaction(41, (const struct sigaction *)&v1, (struct sigaction *)&v4);
return __readfsqword(0x28u) ^ v5;
```

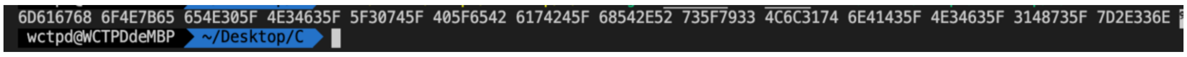
整了很久才高明白程序的流程（诡异的程序调试的时候信号乱发rip乱跳）
大致就是执行到main的时候fork处一个子进程，主进程进入死循环，子进程进行执行，
等子进程执行到401BFC的时候就会按照流程向主进程发送信号，主进程接受信号就会执行相应的函数
所以根据信号流程，这个加密算法应该是这样的：

```
#include <stdio.h>

unsigned int arry[4] = {0x42655f29, 0x9e822efc, 0x0da278c92, 0x4e355a62};
unsigned b = 0x9e3779b9;
int count = 0;
unsigned int c[14];
int main()
{
    unsigned int c1, c2;
    int count1 = 0, v3 = 0;
    char ch[100] = "hgame{No11112121212121212121212121212121212121212121212";
    for (int i = 0; i < 7; i++)
    {
        count1 = 0;
        v3 = count++;
        c1 = *(unsigned int *)(ch + 4 * v3);
        v3 = count++;
        c2 = *(unsigned int *)(ch + 4 * v3);
        for (int j = 0; j < 32; j++)
        {
            c1 += (arry[count1 & 3] + count1) ^ (((c2 >> 5) ^ 16 * c2) + c2);
            count1 += b;
            c2 += (arry[(count1 >> 11) & 3] + count1) ^ (((c1 >> 5) ^ 16 * c1) +
c1);
        }
        c[count - 2] = c1;
        c[count - 1] = c2;
    }
    for (int i = 0; i < 14; i++)
        printf("%X ", c[i]);
}
```

经过查找发现这是xtea加密
解密如下:

```
#include <stdio.h>

int main()
{
    unsigned int array[4] = {0x42655f29, 0x9e822efc, 0x0da278c92, 0x4e355a62};
    unsigned d = 0x9e3779b9;
    unsigned int cipher[14] = {0x27A9C8E9, 0x0BAA973B4, 0x0AAC072F9,
0x0A3FA8000, 0x0D9F4C2D3, 0x0FB3F6BC5, 0x0D3D3D95E,
                               0x86961D77, 0x0E600C53F, 0x98BC27B9, 0x9AAC3AC,
0x6ADC2424, 0x605E304, 0x65E78C77};

    for (int j = 0; j < 14; j+=2)
    {
        int count1 = d << 5;
        for (int i = 0; i < 32; i++)
        {
            cipher[j+1] -= (array[(count1 >> 11) & 3] + count1) ^ (((cipher[j]
>> 5) ^ 16 * cipher[j]) + cipher[j]);
            count1 -= d;
            cipher[j] -= (array[count1 & 3] + count1) ^ (((cipher[j+1] >> 5) ^
16 * cipher[j+1]) + cipher[j+1]);
        }
    }
    for (int i = 0; i < 14; i++)
        printf("%X ", cipher[i]);
    return 0;
}
```

得到一串16进制



```
6D616768 6F4E7B65 654E305F 4E34635F 5F30745F 405F6542 6174245F 68542E52 735F7933 4C6C3174 6E41435F 4E34635F 3148735F 7D2E336E
wctpd@WCTPDdeMBP ~/Desktop/C
```

再把每一串16进制倒过来转成字符串就是flag了