Week4　write up

RE

0x01 Happy VM

看了学习资料之后,大概了解了 VM 这种题型
可以看到
初始化了六个寄存器和栈

```
__int64 sub_4007B5()
{
  __int64 result; // rax

  stack = calloc(1uLL, 0x100uLL);
  S = 0LL;
  R4 = 0;
  R3 = 0;
  R2 = 0;



  R1 = 0;
  result = 0LL;
  R0 = 0;
  R5 = 0;
  return result;
}
```

数组是对应的汇编指令

```
char vopcode[]= {0x11,0x2D,0x00,0x22,0x05,0x10,0x14,0x09,
                 0x17,0x00,0x32,0x05,0x03,0x11,0x16,0x06,
                 0x00,0x16,0x05,0x11,0x16,0x17,0x0E,0x01,
                 0x15,0x04,0x0F,0x01,0x16,0x02,0x00,0x00,
                 0x04,0x03,0x05,0x10,0x14,0x2B,0x05,0x09,
                 0x03,0x13,0x16,0x05,0x12,0x15,0x04,0x10,
                 0x14,0x36,0x0A,0x01,0x13,0x2D,0x03,0x04,0x12};
```

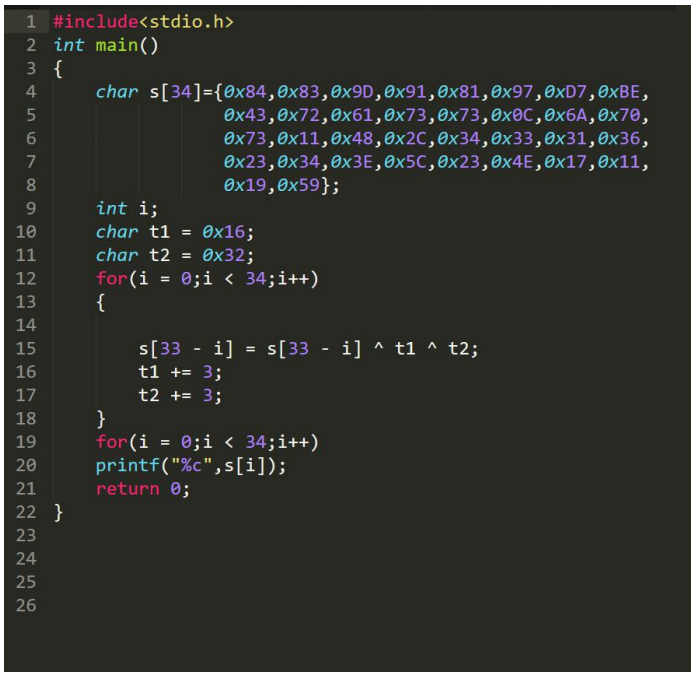
函数 VM 是汇编指令的解释器

```
__int64 __fastcall VM(unsigned __int64 p1, __int64 p2)
{
  char v2; // al
  char v3; // al
  __int64 result; // rax
  char *input_; // [rsp+0h] [rbp-20h]
  char parameter; // [rsp+1Eh] [rbp-2h]
  char v7; // [rsp+1Fh] [rbp-1h]

  input_ = p2;
  S = p2;
  parameter = 0;
  while ( 1 )
  {
    v3 = R4++;
    result = *(v3 + p1);
    v7 = result;
    if ( result == 0x17 )
      return result;
    switch ( result )
    {
      case 0u:
      case 8u:
      case 9u:
      case 0xAu:
      case 0xCu:
      case 0xDu:
      case 0xEu:
      case 0x11u:
      case 0x13u:
      case 0x14u:
        v2 = R4++;
        parameter = *(v2 + p1);
        break;
00000856 VM:17 (400856)
```

```
push 2
jmp  label 0
push 0x22
pop  reg1
cmp  reg0 reg1 //bool
je   label $09
end
.label $09
push $0x32 // reg0 == reg1 == reg2 == 0x22
pop  reg1  // reg1 = 0x32
push reg2 // 0x22
push $0f
jmp  label $16
pop  reg2 // reg2 = 0x22
push $16
pop  reg1 // reg1 = 0x16
push $15
jmp  label$16
end
.label $16
sub  reg2  $0x01
push s[reg2]
pop  reg0
xor  reg0  reg1
push reg0
pop  s[reg2]    ***
push reg1
push $0x00
pop  reg0
push reg2
.label $22
pop  reg1
cmp  reg0  reg1
je   label $2b
pop  reg1
add  reg1  $0x3
jmp  label $16
.label $2b
pop  reg1
pop  pN
jmp  pN
.label $2d
push s[reg2]
pop  reg0
cmp  reg0  reg1
je   label $36
add  reg2  $01
jmp  label $2d
.label $36
push reg2
pop  reg0
pop  reg4
jmp  reg4
```

这段代码就是对 flag 进行了两次异或操作
接着写出逆算法就 ok 了



```c
#include<stdio.h>
int main()
{
    char s[34]={0x84,0x83,0x9D,0x91,0x81,0x97,0xD7,0xBE,
                0x43,0x72,0x61,0x73,0x73,0x0C,0x6A,0x70,
                0x73,0x11,0x48,0x2C,0x34,0x33,0x31,0x36,
                0x23,0x34,0x3E,0x5C,0x23,0x4E,0x17,0x11,
                0x19,0x59};
    int i;
    char t1 = 0x16;
    char t2 = 0x32;
    for(i = 0;i < 34;i++)
    {

        s[33 - i] = s[33 - i] ^ t1 ^ t2;
        t1 += 3;
        t2 += 3;
    }
    for(i = 0;i < 34;i++)
    printf("%c",s[i]);
    return 0;
}
```

Flag:hgame{3Z_VM_W0NT_5T0P_UR_PR0GR355}