Real

程序在 init 里面调用释放.real.so 并执行,real.so 第一次执行 puts 函数时对自身 encrypt 函数进行修改:

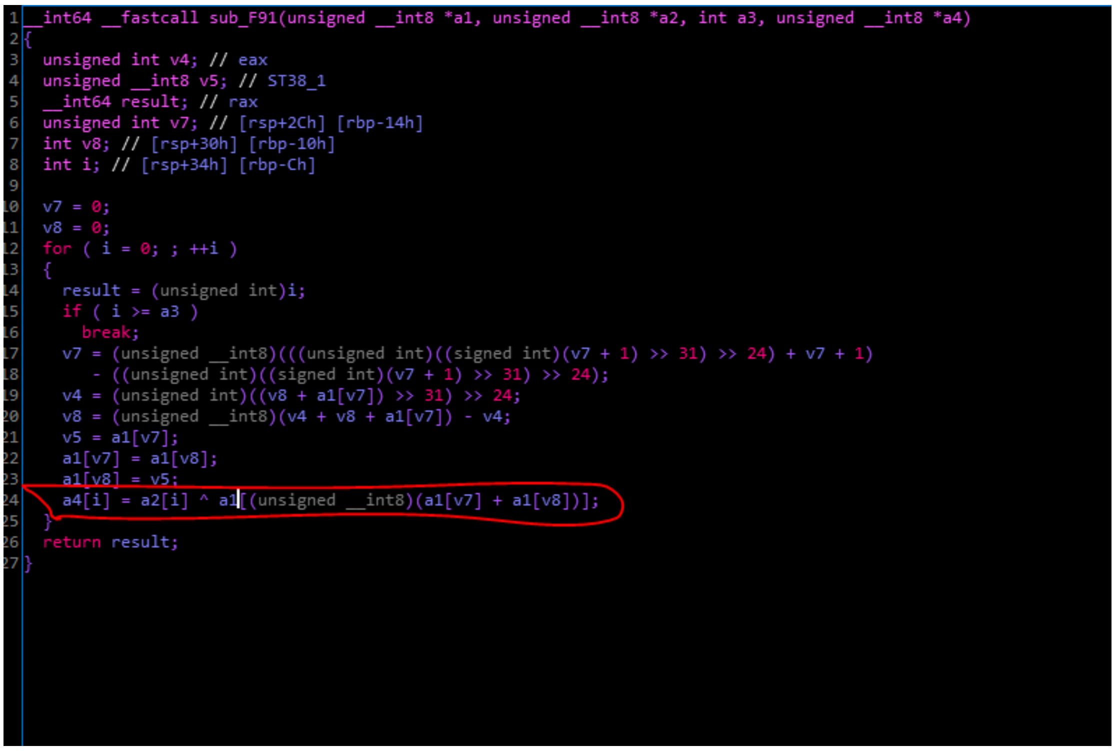
```
for ( i = 0; i <= 309; ++i )
    *((_BYTE *)encrypt + i + 20) ^= i;
```

解密后的代码差不多是

```
memset(v6, 0, sizeof(v6));
memset(s2, 0, sizeof(s2));
scanf("%50s", v6, s2, a1);
v1 = strlen("hgame!@#");
unk_EB1(v2, "hgame!@#", v1);
v3 = strlen(v6);
unk_F91(v2, v6, v3, s2);
if ( !strcmp(s1, s2) )
    printf("success!", s2);
else
    printf("failed", s2);
putchar(10);
return __readfsqword(0x28u) ^ v8;
```

其中 unk_EB1 参数只有长度是变量，根据 strcmp 参数 s1 的长度推测出字符串长度 27

unk_F91 对 s2 的修改



```
__int64 __fastcall sub_F91(unsigned __int8 *a1, unsigned __int8 *a2, int a3, unsigned __int8 *a4)
{
  unsigned int v4; // eax
  unsigned __int8 v5; // ST38_1
  __int64 result; // rax
  unsigned int v7; // [rsp+2Ch] [rbp-14h]
  int v8; // [rsp+30h] [rbp-10h]
  int i; // [rsp+34h] [rbp-Ch]

  v7 = 0;
  v8 = 0;
  for ( i = 0; ; ++i )
  {
    result = (unsigned int)i;
    if ( i >= a3 )
      break;
    v7 = (unsigned __int8)(((unsigned int)((signed int)(v7 + 1) >> 31) >> 24) + v7 + 1)
       - ((unsigned int)((signed int)(v7 + 1) >> 31) >> 24);
    v4 = (unsigned int)((v8 + a1[v7]) >> 31) >> 24;
    v8 = (unsigned __int8)(v4 + v8 + a1[v7]) - v4;
    v5 = a1[v7];
    a1[v7] = a1[v8];
    a1[v8] = v5;
    a4[i] = a2[i] ^ a1[(unsigned __int8)(a1[v7] + a1[v8])];
  }
  return result;
}
```

只有这一处 改成

```
a2[i]= a4[i] ^ a1[(unsigned __int8)(a1[v7] + a1[v8])];
```

就可以计算出 flag

happyVM

先把 vm 的代码稍微的列出来，

```
\x11\x2D         push nextIns    currentLength=0x2d
\x00\x22         push 0x22
\x05             pop tmpcode2
\x10             byte_602085= tmpcode1==tmpcode2
\x14\x09         if byte_602085 currentLength=0x8
\x17             nop
-----0x8-----
\x00\x32         push 0x32
\x05             pop tmpcode2
\x03             push tmpcode3
\x11\x16         push nextIns currentLength=0x16
\x06             pop tmpcode3
\x00\x16         push 0x16
\x05             pop tmpcode2
\x11\x16         push nextIns currentLength=0x16
\x17             nop
-----0x16-----
\x0E\x01         tmpcode3-=0x1

\x15             push outflag[tmpcode3]
\x04             pop tmpcode1

\x0F             tmpcode1^=tmpcode2

\x01             push tmpcode1
\x16             pop outflag[tmpcode3]

\x02             push tmpcode2

\x00\x00         push 0
\x04             pop tmpcode1
\x03             push tmpcode3
\x05             pop tmpcode2

\x10             byte_602085=tmpcode1==tmpcode2
\x14\x2B         if byte_602085 currentLength=0x2b
\x05             pop tmpcode2
\x09\x03         tmpcode2+=0x3
\x13\x16         currentLength=0x16
-----0x2b-----
\x05             pop tmpcode2
\x12             pop currentLength
-----2d-----
```

```
\x15            push outflag[tmpcode3]
\x04            pop tmpcode1
\x10            byte_602085=   tmpcode1== tmpcode2
\x14\x36        if byte_602085 currentLength=0x36
\x0A\x01        tmpcode3+=1
\x13\x2D        currentLength=0x2d
\x03            push tmpcode3
\x04            pop tmpcode1
\x12            pop currentLength
```

差不多是有一个栈的结构

然后，把他人工优化

```
\x11\x2D        push nextIns   jmp=0x2d
\x00\x22        tmpcode2 = 0x22
\x10            byte_602085= tmpcode1==tmpcode2
\x14\x09        if byte_602085 jmp=0x8
\x17            nop
-----0x8-----
\x00\x32        tmpcode2= 0x32
\x03            push tmpcode3
\x11\x16        push nextIns jmp=0x16
\x06            pop tmpcode3
\x00\x16        push 0x16
\x05            pop tmpcode2
\x11\x16        push nextIns jmp=0x16
\x17            nop
-----0x16-----
\x0E\x01        tmpcode3-=0x1
\x0F            outflag[tmpcode3]^=tmpcode2
\x00\x00        tmpcode1=0
\x10            byte_602085= tmpcode1== tmpcode3
\x14\x2B        if byte_602085 jmp=0x2b
\x09\x03        tmpcode2+=0x3
\x13\x16        jmp=0x16
-----0x2b-----
\x05            pop tmpcode2
\x12            pop currentLength
-----2d-----
\x04            tmpcode1=outflag[tmpcode3]
\x10            byte_602085=   tmpcode1== tmpcode2
\x14\x36        if byte_602085 jmp=0x36
\x0A\x01        tmpcode3+=1
\x13\x2D        jmp=0x2d

\x04            tmpcode1=tmpcode3
```

\x12   pop currentLength

在继续人工 f5，就可以看出是 2 个异或操作
下面是解密程序

```
char a[] = {
"\x84\x83\x9D\x91\x81\x97\xD7\xBE\x43\x72\x61\x73\x73\x0C\x6A\x70"
"\x73\x11\x48\x2C\x34\x33\x31\x36\x23\x34\x3E\x5C\x23\x4E\x17\x11"
"\x19\x59\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
"\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00"
};

int main()
{
    int aaa = 0x16;

    for (int i = 0x21; i >= 0; --i)
    {
        a[i] = a[i] ^ aaa;
        aaa += 0x3;
    }
    aaa = 0x32;
    for (int i = 0x21; i >=0; --i)
    {
        a[i] = a[i] ^ aaa;
        aaa += 0x3;
    }
    cout << a;
}
```