一次不太理想的逆向之simple-vm

原创

飞鸿踏雪（蓝屏选手） 于 2019-08-25 20:41:00 发布 171 收藏

分类专栏： 逆向 文章标签： ctf、reverse

本文链接：https://blog.csdn.net/qq_41252520/article/details/100066822

前言

- 之所以说是不太理想，不是因为解不出来 **flag** ，而是部分流程不明确，最后的结果也是猜测出来的。尽管有相关的 **writeup** 但是对于里面提到的一些关键信息，我未能从反编译结果中获取。

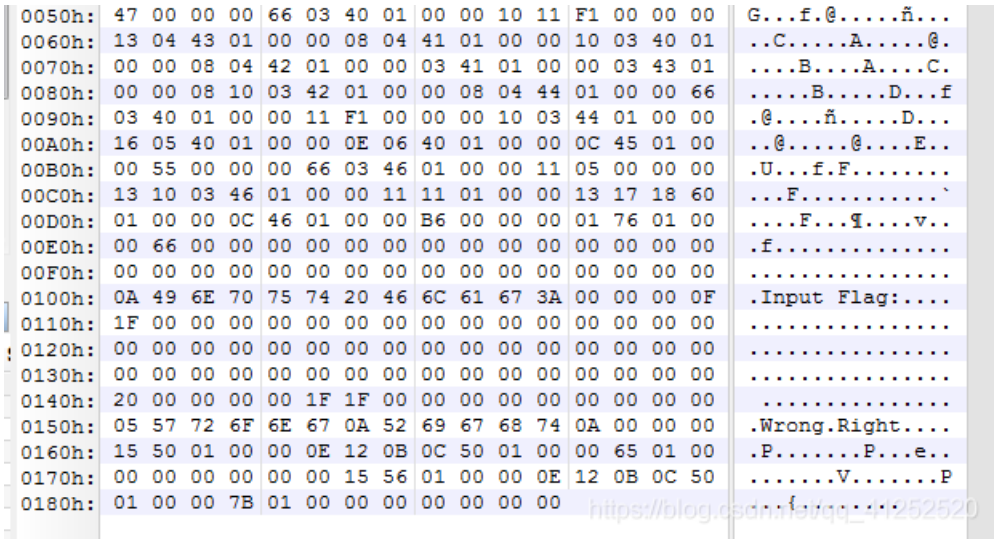
分析

```
  void *v8; // rax

  fin = fopen("p.bin", "rb");
  v4 = "err 0";
  if ( !fin )
    goto LABEL_4;
  v5 = fin;
  fseek(fin, 0LL, 2);
  file_size = ftell(v5);
  fseek(v5, 0LL, 0);
  v6 = file_size;
  if ( file_size <= 0 )
  {
    v4 = "err 1";
LABEL_4:
    puts(v4);
    return 0xFFFFFFFFLL;
  }
  v8 = malloc(file_size);
  ptr = v8;
  v4 = "err 3";
  if ( !v8 )
    goto LABEL_4;
  v4 = "err 4";
  if ( file_size != fread(v8, 1uLL, v6, v5) )
    goto LABEL_4;
  fclose(v5);
  v4 = "err 5";
  if ( (unsigned int)VM_Start() )
    goto LABEL_4;
  free(ptr);
  return 0LL;
}
```


打开同目录下的文件，读取数据到 **ptr** 中，开始虚拟机。先看一下 **bin** 文件

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0123456789ABCDEF
0000h: 01 30 00 00 00 10 18 43 14 15 47 40 17 10 1D 4B .0.....C..G@...K
0010h: 12 1F 49 48 18 53 54 01 57 51 53 05 56 5A 08 58 ..IH.ST.WQS.VZ.X
0020h: 5F 0A 0C 58 09 00 01 02 03 04 05 06 00 00 00 00 _..X............
0030h: 15 00 01 00 00 0E 12 0B 0C 00 01 00 00 35 00 00 .............5..
0040h: 00 66 15 10 01 00 00 0E 0A 66 16 0C 10 01 00 00 .f.......f......
```



```
0050h: 47 00 00 00 66 03 40 01 00 00 10 11 F1 00 00 00  G...f.@.....ñ...
0060h: 13 04 43 01 00 00 08 04 41 01 00 00 10 03 40 01  ..C.....A.....@.
0070h: 00 00 08 04 42 01 00 00 03 41 01 00 00 03 43 01  ....B....A....C.
0080h: 00 00 08 10 03 42 01 00 00 08 04 44 01 00 00 66  .....B.....D...f
0090h: 03 40 01 00 00 11 F1 00 00 00 10 03 44 01 00 00  .@....ñ.....D...
00A0h: 16 05 40 01 00 00 0E 06 40 01 00 00 0C 45 01 00  ..@.....@....E..
00B0h: 00 55 00 00 00 66 03 46 01 00 00 11 05 00 00 00  .U...f.F........
00C0h: 13 10 03 46 01 00 00 11 11 01 00 00 13 17 18 60  ...F...........`
00D0h: 01 00 00 0C 46 01 00 00 B6 00 00 00 01 76 01 00  ....F...¶....v..
00E0h: 00 66 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .f..............
00F0h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0100h: 0A 49 6E 70 75 74 20 46 6C 61 67 3A 00 00 00 0F  .Input Flag:....
0110h: 1F 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0120h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0130h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0140h: 20 00 00 00 00 1F 1F 00 00 00 00 00 00 00 00 00   ...............
0150h: 05 57 72 6F 6E 67 0A 52 69 67 68 74 0A 00 00 00  .Wrong.Right....
0160h: 15 50 01 00 00 0E 12 0B 0C 50 01 00 00 65 01 00  .P.......P...e..
0170h: 00 00 00 00 00 00 15 56 01 00 00 0E 12 0B 0C 50  .......V.......P
0180h: 01 00 00 7B 01 00 00 00 00 00 00 00              ...{........
```


再来关注 **VM_Start** 函数。

```
__int64 sub_400896()
{
  __int64 v0; // rax
  _BYTE *v1; // rbp
  int index; // ebx
  __int64 v4; // rdx
  __int64 v5; // rax
  __int64 v6; // rax
  __int64 v7; // rax
  __int64 v8; // rax
  __int64 v9; // rax
  int v10; // eax
  __int64 v11; // rax
  char v12; // dl
  int v13; // eax
  int v14; // eax
  _BYTE *input_len; // rax
  __int64 v16; // rax
  __int64 v17; // rax
  __int64 v18; // rax

  v0 = 0LL;
  v1 = ptr;
  while ( 1 )
  {
    index = v0 + 1;
    switch ( v1[v0] )
    {
      case 0:
        return *(unsigned int *)&v1[index];
      case 1:
        goto LABEL_35;
      case 2:
        v4 = index;
        index = v0 + 9;
        v1[*(signed int *)&v1[v4]] = *(_DWORD *)&v1[(signed int)v0 + 5];
        break;
      case 3:
        v5 = index;
        index += 4;
```

```
        v6 = *(signed int *)&v1[v5];
        goto LABEL_27;
      case 4:
        v7 = index;
        index += 4;
        v8 = *(signed int *)&v1[v7];
        goto LABEL_31;
      case 5:
        v9 = index;
        index += 4;
        v10 = (char)v1[*(signed int *)&v1[v9]];
        goto LABEL_21;
      case 6:
        v11 = index;
        v12 = tmp1;
        index += 4;
        v8 = *(signed int *)&v1[v11];
        goto LABEL_9;
      case 7:
        v13 = tmp1;
        goto LABEL_23;
      case 8:
        v14 = ~(tmp1 & tmp2);
        goto LABEL_12;
      case 0xA:                                 // 读取用户输入的一个字节
        v14 = getchar();
        goto LABEL_12;
      case 0xB:
        putchar(tmp2);
        break;
      case 0xC:                                 // 计算字符串剩余长度
        input_len = &v1[*(signed int *)&v1[index]];
        if ( *input_len )
        {
          index = *(_DWORD *)&v1[index + 4];
          --*input_len;
        }
        else
        {
          index += 8;
        }
        break;
      case 0xD:
        ++tmp2;
        break;
      case 0xE:
        ++tmp1;
        break;
      case 0xF:
        v14 = tmp1;
        goto LABEL_12;
      case 0x10:
        v10 = tmp2;
        goto LABEL_21;
      case 0x11:
        v16 = index;
        index += 4;
        v13 = *(_DWORD *)&v1[v16];
LABEL_23:
        tmp2 += v13;
```

```
        break;
      case 0x12:
        v6 = tmp1;
        goto LABEL_27;
      case 0x13:
        v6 = tmp2;
LABEL_27:
        v14 = (char)v1[v6];
        goto LABEL_12;
      case 0x14:
        v17 = index;
        index += 4;
        v14 = *(_DWORD *)&v1[v17];
        goto LABEL_12;
      case 0x15:
        v18 = index;
        index += 4;
        v10 = *(_DWORD *)&v1[v18];
LABEL_21:
        tmp1 = v10;
        break;
      case 0x16:
        v8 = tmp1;
LABEL_31:
        v12 = tmp2;
LABEL_9:
        v1[v8] = v12;
        break;
      case 0x17:
        v14 = tmp2 - tmp1;
LABEL_12:
        tmp2 = v14;
        break;
      case 0x18:
        if ( tmp2 )
LABEL_35:
          index = *(_DWORD *)&v1[index];
        else
          index = v0 + 5;
        break;
      default:
        break;
    }
    if ( index >= file_size )
      return 0LL;
    v0 = index;
  }
}
```

动态调试，可以得知 输入的字符串长度为 **0x20** 。

```
0000000001E3B340  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000001E3B350  00 49 6E 70 75 74 20 46  6C 61 67 3A 00 00 00 0F  .Input·Flag:....
0000000001E3B360  1F 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
0000000001E3B370  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

字符串长度

* 输入的数据存放在下个地址处

```
0000000001E3B350  00 49 6E 70 75 74 20 46  6C 61 67 3A 00 00 00 0F  .Input·Flag:....
```

```
0000000001E3B360  10 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61  .aaaaaaaaaaaaaaa
0000000001E3B370  61 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  a...............
0000000001E3B380  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

本程序结构比较简单，为了方便观察程序的流程，我们将他的代码重构一下，放到 **VS** 中进行调试。

重构代码

```
#include<stdio.h>
#include<stdlib.h>
#include"defs.h"
int file_size;
void *ptr;
__int64 sub_400896()
{
 __int64 v0; // rax
 unsigned char *v1; // rbp
 int index; // ebx
 __int64 v4; // rdx
 __int64 v5; // rax
 __int64 v6; // rax
 __int64 v7; // rax
 __int64 v8; // rax
 __int64 v9; // rax
 int v10; // eax
 __int64 v11; // rax
 char v12; // dl
 int v13; // eax
 int v14; // eax
 unsigned char *input_len; // rax
 __int64 v16; // rax
 __int64 v17; // rax
 __int64 v18; // rax
 int tmp1, tmp2;
 tmp1 = tmp2 = 0;
 v0 = 0LL;
 v1 = (unsigned char*)ptr;
 while (1)
 {
  index = v0 + 1;
  switch (v1[v0])
  {
  case 0:
   return *(unsigned int *)&v1[index];
  case 1:
   goto LABEL_35;
  case 2:
   //printf("index= 0x%x\n", index);
   v4 = index;
   //printf("v4 = index;\n");
   //printf("v4= 0x%x\n", v4);
   //printf("v0= 0x%x\n", v0);
   index = v0 + 9;
   //printf("index = v0 + 9;\n");
   //printf("index= 0x%x\n", index);
   v1[*(signed int *)&v1[v4]] = *(unsigned int *)&v1[(signed int)v0 + 5];
   printf("v1[*(signed int *)&v1[v4]] = *(unsigned int *)&v1[(signed int)v0 + 5];\n");
   printf("v1[*(signed int *)&v1[v4]] = 0x%x;\n", *(unsigned int *)&v1[(signed int)v0 + 5]);
```

```
 printf("v1[*(signed int *)&v1[v4]] = 0x%x;\n", *(unsigned int *)&v1[(signed int)v0 + 3]);
 break;
case 3:
 //printf("index= 0x%x\n", index);
 v5 = index;
 //printf("v5 = index;\n");
 //printf("v5= 0x%x\n", v5);
 index += 4;
 //printf("index += 4;\n");
 //printf("index= 0x%x\n", index);
 v6 = *(signed int *)&v1[v5];
 //printf("v6 = *(signed int *)&v1[v5];\n");
 //printf("v6= 0x%x\n", v6);
 goto LABEL_27;
case 4:
 //printf("index= 0x%x\n", index);
 v7 = index;
 //printf("v7 = index;\n");
 //printf("v7= 0x%x\n", v7);
 index += 4;
 //printf("index += 4;\n");
 //printf("index= 0x%x\n", index);
 v8 = *(signed int *)&v1[v7];
 //printf("v8 = *(signed int *)&v1[v7];\n");
 //printf("v8= 0x%x\n", v8);
 goto LABEL_31;
case 5:
 //printf("index= 0x%x\n", index);
 v9 = index;
 //printf("v9 = index;\n");
 //printf("v9= 0x%x\n", v9);
 index += 4;
 //printf("index += 4;\n");
 //printf("index= 0x%x\n", index);
 v10 = (char)v1[*(signed int *)&v1[v9]];
 printf("v10 = (char)v1[*(signed int *)&v1[v9]];\n");
 printf("v10= 0x%x\n", v10);
 goto LABEL_21;
case 6:
 //printf("index= 0x%x\n", index);
 v11 = index;
 //printf("v11 = index;\n");
 //printf("v11= 0x%x\n", v11);
 v12 = tmp1;
 //printf("v12 = tmp1;\n");
 //printf("v12= 0x%x\n", v12);
 index += 4;
 //printf("index += 4;\n");
 //printf("index= 0x%x\n", index);
 v8 = *(signed int *)&v1[v11];
 //printf("v8 = *(signed int *)&v1[v11];\n");
 //printf("v8= 0x%x\n", v8);
 goto LABEL_9;
case 7:
 //printf("v13 = tmp1;\n");
 v13 = tmp1;
 //printf("v13= 0x%x\n",tmp1);
 goto LABEL_23;
case 8:
 printf("tmp1= 0x%x tmp2= 0x%x\n",tmp1,tmp2);
```

```
 v14 = ~(tmp1 & tmp2);
 printf("v14 = ~(tmp1 & tmp2);\n");
 printf("v14= 0x%x\n", v14);
 goto LABEL_12;
case 0xA:                               // 读取用户输入的一个字节
 v14 = getchar();
 goto LABEL_12;
case 0xB:
 printf("\n");
 putchar(tmp2);
 printf("\n");
 break;
case 0xC:                               // 计算字符串剩余长度
 input_len = &v1[*(signed int *)&v1[index]];
 if (*input_len)
 {
  index = *(unsigned int *)&v1[index + 4];
  --*input_len;
 }
 else
 {
  index += 8;
 }
 break;
case 0xD:
 ++tmp2;
 printf("++tmp2\n");
 printf("tmp2= 0x%x\n", tmp2);
 break;
case 0xE:
 ++tmp1;
 printf("++tmp1\n");
 printf("tmp1= 0x%x\n", tmp1);
 printf("\n");
 break;
case 0xF:
 v14 = tmp1;
 //("v14 = tmp1;\n");
 //printf("v14 = 0x%x\n", tmp1);
 goto LABEL_12;
case 0x10:
 v10 = tmp2;
 printf("v10 = tmp2;\n");
 printf("v10 = 0x%x\n", tmp2);
 goto LABEL_21;
case 0x11:
 //printf("index= 0x%x\n", index);
 v16 = index;
 //printf("v16 = index;\n");
 //printf("v16= 0x%x\n", index);
 index += 4;
 //printf("index += 4;\n");
 //printf("index= 0x%x\n", index);
 v13 = *(unsigned int *)&v1[v16];
 //printf("v13 = *(unsigned int *)&v1[v16];\n");
 //printf("v13= 0x%x\n", v13);
LABEL_23:
 //printf("tmp2 = 0x%x v13= 0x%x\n",  tmp2,v13);
 tmp2 += v13;
 //printf("tmp2 += v13;\n");
```

```
 //printf("tmp2 += v13;\n");
 //printf("tmp2 = 0x%x;\n",tmp2);
 break;
case 0x12:
 v6 = tmp1;
 //printf("v6 = tmp1;\n");
 //printf("v6 = 0x%x\n",v6);
 goto LABEL_27;
case 0x13:
 v6 = tmp2;
 //printf("v6 = tmp2;\n");
 //printf("v6 = 0x%x\n", v6);
LABEL_27:
 v14 = (char)v1[v6];
 //printf("v14 = (char)v1[v6];\n");
 //printf("v14 = 0x%x\n",v14);
 goto LABEL_12;
case 0x14:
 //printf("index= 0x%x\n", index);
 v17 = index;
 //printf("v17 = index;\n");
 //printf("v17= 0x%x\n", v17);
 index += 4;
 //printf("index += 4;\n");
 //printf("index= 0x%x\n", index);
 v14 = *(unsigned int *)&v1[v17];
 //printf("v14 = *(unsigned int *)&v1[v17];\n");
 //printf("v14= 0x%x\n", v14);
 goto LABEL_12;
case 0x15:
 //printf("index= 0x%x\n", index);
 v18 = index;
 //printf("v18 = index;\n");
 //printf("v18= 0x%x\n", v18);
 index += 4;
 //printf("index += 4;\n");
 //printf("index= 0x%x\n", index);
 v10 = *(unsigned int *)&v1[v18];
 printf("v10 = *(unsigned int *)&v1[v18];\n");
 printf("v10= 0x%x\n", v10);
LABEL_21:
 tmp1 = v10;
 printf("tmp1 = v10;\n");
 break;
case 0x16:
 v8 = tmp1;
 //printf("op:0x16 v8= 0x%x\n", tmp1);
LABEL_31:
 v12 = tmp2;
 printf("v12 = tmp2;\n");
 //printf("v12= 0x%x\n", tmp2);
LABEL_9:
 v1[v8] = v12;
 printf("v1[v8] = v12;\n");
 printf("v8= 0x%x v12= 0x%x;\n",v8,v12);
 break;
case 0x17:
 printf("tmp2= 0x%x tmp1= 0x%x\n",tmp2,tmp1);
 v14 = tmp2 - tmp1;
 printf("v14 = tmp2 - tmp1;\n");
```

```
   printf("v14= 0x%x\n", v14);
  LABEL_12:
   printf("tmp2 = v14;\n");
   tmp2 = v14;
   //printf("tmp2 = 0x%x\n", v14);
   break;
  case 0x18:
   if (tmp2) {
   LABEL_35:
    printf("index= 0x%x\n", index);
    index = *(unsigned int *)&v1[index];
    printf("index = *(unsigned int *)&v1[index];");
    printf("index= 0x%x\n", index);
   }
   else {
    printf("v0= 0x%x\n", v0);
    index = v0 + 5;
    printf("index = v0 + 5;\n");
    printf("index= 0x%x\n", index);
   }
   break;
  default:
   break;
  }
  if (index >= file_size)
   return 0LL;
  v0 = index;
 }
}
int main(__int64 a1, char **a2, char **a3)
{
 FILE *fin; // rax
 const char *v4; // rdi
 FILE *v5; // rbx
 size_t v6; // rbp
 void *v8; // rax

 fin = fopen("p.bin", "rb");
 v4 = "err 0";
 if (!fin)
  goto LABEL_4;
 v5 = fin;
 fseek(fin, 0LL, 2);
 file_size = ftell(v5);
 fseek(v5, 0LL, 0);
 v6 = file_size;
 if (file_size <= 0)
 {
  v4 = "err 1";
 LABEL_4:
  puts(v4);
  return 0xFFFFFFFFLL;
 }
 v8 = malloc(file_size);
 ptr = v8;
 v4 = "err 3";
 if (!v8)
  goto LABEL_4;
 v4 = "err 4";
```

```
if (file_size != fread(v8, 1uLL, v6, v5))
 goto LABEL_4;
fclose(v5);
v4 = "err 5";
if ((unsigned int)sub_400896())
 goto LABEL_4;
getchar();
free(ptr);
return 0LL;
}
```

将程序的输出保存到文件中，观察日志，发现一些规律：

```
...
tmp1= 0x20 tmp2= 0x61
v14 = ~(tmp1 & tmp2);
v14= 0xffffffdf
...
++tmp1
tmp1= 0x21
...
tmp1= 0x21 tmp2= 0x61
v14 = ~(tmp1 & tmp2);
v14= 0xffffffde
...
```

注意：~(x & y) <=> x xor y

我们想要知道在什么条件下才能使得程序输出 **Right**，看到日志的后面

```
...
tmp1= 0xffffffbf tmp2= 0xfffffffe1
v14 = ~(tmp1 & tmp2);
v14= 0x5e
tmp2 = v14;
v12 = tmp2;
v1[v8] = v12;
v8= 0x144 v12= 0x0;
tmp2 = v14;
v10 = tmp2;
v10 = 0x130
tmp1 = v10;
tmp2 = v14;
v12 = tmp2;
v1[v8] = v12;
v8= 0x130 v12= 0x0;
v10 = (char)v1[*(signed int *)&v1[v9]];
v10= 0x3f
tmp1 = v10;
++tmp1
tmp1= 0x40

v1[v8] = v12;
v8= 0x140 v12= 0x0;
tmp2 = v14;
tmp2 = v14;
v10 = tmp2;
v10 = 0x9
tmp1 = v10;
```

```
tmp2 = v14;
tmp2= 0x5e tmp1= 0x9
v14 = tmp2 - tmp1;
v14= 0x55
tmp2 = v14;
...
```

结果当然是输出 **Wrong** ， 而当我们将

```
if (tmp2) {
   LABEL_35:
```

改为

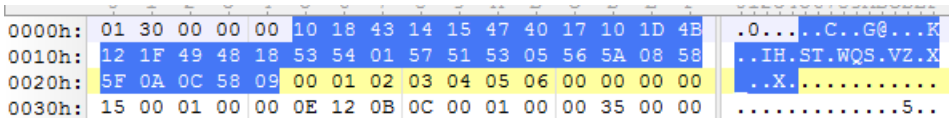
```
if (0) {
   LABEL_35:
```

无论输入什么程序都输出 **Right** ，看来只有当最后 **tmp2= 0** 时，程序才输出 **Right** 。

可就这个条件，也只能求得最后一个字符为 **6**，前面的 **31** 个字符没有办法求出来。观察日志，从文件中找到 **9** 所在的位置。猜测程序的算法是

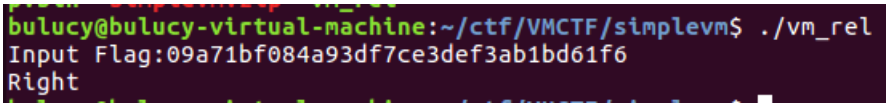
```
for(int i=0;i<0x20;i++){
 ret=input[i]^(0x20+i);
}
```

input 由用户输入，最后得到的 **ret** 与文件中对应的数据进行比较。从文件中找到这 **32** 个数据



脚本

```
data=[0x10,0x18,0x43,0x14,0x15,0x47,0x40,0x17,0x10,0x1d,0x4b,0x12,0x1f,0x49,0x48,0x18,0x53,0x54,0x1,0x57,0x51,0x53,0x5,0x56,0x5a,0x8,0x58,0x5f,0xa,0xc,0x58,0x9]
flag=''
for i in range(len(data)):
    flag+=chr(data[i]^0x20+i)
print flag
```



- flag：09a71bf084a93df7ce3def3ab1bd61f6
- 就这样？Em…尽管如此，也花去了我一天的时间。o(╥﹏╥)o

总结

对于结构简单的 VM，可以重构代码，从源码级的角度去调试，输出每条执行的指令以及数据以便弄清楚程序的流程。

逆向 VM真的需要很大的耐心和毅力，不要害怕花费时间。

然而 会编写**Pintool**脚本的话只用 **20** 分钟就搞定了，奈何我不会啊，得去学习学习~

参考 虚拟机保护逆向入门