IDF-CTF-简单的Elf逆向Writeup

转载

baikeng3674 于 2017-02-04 17:37:00 发布 349 收藏 1
文章标签： python php
原文链接：http://www.cnblogs.com/WangAoBo/p/6365987.html

ElfCrackMe1

简单的Elf逆向Writeup

> 题目来源：IDF实验室 CTF训练营；题目链接http://ctf.idf.cn/index.php?g=game&m=article&a=index&id=39
> 题目下载：http://pan.baidu.com/s/1kTl5wxD

解法1：

IDA查看伪代码法：

1. 下载文件，现在Linux环境下运行，可以看到关键字符串*u r wrong*和*plz enter the flag:*
2. 把文件拖到IDA中，shift+F12查找字符串，双击*u r right*跳转到相应位置
3. 如下图，双击调用关键字符串*u r right*的函数，
   跳转后F5查看伪代码如下

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  int v3; // ebx@6
  const char **v4; // rdx@22
  __int64 v6; // [sp+0h] [bp-C0h]@1
  __int64 v7; // [sp+8h] [bp-B8h]@1
  __int64 v8; // [sp+10h] [bp-B0h]@1
  __int64 v9; // [sp+18h] [bp-A8h]@1
  __int64 v10; // [sp+20h] [bp-A0h]@1
  __int64 v11; // [sp+28h] [bp-98h]@1
  __int64 v12; // [sp+30h] [bp-90h]@1
  __int64 v13; // [sp+38h] [bp-88h]@1
  int v14; // [sp+40h] [bp-80h]@1
  char v15[17]; // [sp+80h] [bp-40h]@2
  char v16; // [sp+91h] [bp-2Fh]@14
  char v17; // [sp+92h] [bp-2Eh]@15
  char v18; // [sp+93h] [bp-2Dh]@16
  char v19; // [sp+94h] [bp-2Ch]@17
  char v20; // [sp+95h] [bp-2Bh]@18
  int v21; // [sp+A4h] [bp-1Ch]@1
  int v22; // [sp+A8h] [bp-18h]@7
  int i; // [sp+ACh] [bp-14h]@9

  v21 = 0;
  memset(&v6, 0, 0x58uLL);
  v6 = 854698492143LL;
  v7 = 880468295913LL;
  v8 = 597000454391LL;
  v9 = 605590388953LL;
```

```
30  v10 = 932007903423LL;
31  v11 = 760209211613LL;
32  v12 = 579820585151LL;
33  v13 = 940597838039LL;
34  v14 = 191;
35  printf(
36    "plz enter the flag:",
37    argv,
38    11LL,
39    854698492143LL,
40    880468295913LL,
41    597000454391LL,
42    605590388953LL,
43    932007903423LL,
44    760209211613LL,
45    579820585151LL,
46    940597838039LL,
47    *(_QWORD *)&v14);
48  while ( 1 )
49  {
50    v3 = v21;
51    v15[v3] = getch();
52    if ( !v15[v3] || v15[v21] == 10 )
53      break;
54    if ( v15[v21] == 8 )
55    {
56      printf("\b\b", v6, v7, v8, v9, v10, v11, v12, v13, *(_QWORD *)&v14);
57      --v21;
58    }
59    else
60    {
61      putchar(v15[v21++]);
62    }
63  }
64  v22 = 0;
65  if ( v21 != 22 )
66    v22 = 1;
67  for ( i = 0; i <= 16; ++i )
68  {
69    if ( v15[i] != (*((_DWORD *)&v6 + i) - 1) / 2 )
70    {
71      v22 = 1;
72      argv = (const char **)((*((_DWORD *)&v6 + i) - 1) / 2);
73      printf("%d", argv, v6, v7, v8, v9, v10, v11, v12, v13, *(_QWORD *)&v14);
74    }
75  }
76  if ( v16 != 48 || v17 != 56 || v18 != 50 || v19 != 51 || v20 != 125 )
77    v22 = 1;
78  v15[v21] = 0;
79  puts("\r");
80  if ( v22 )
81  {
82    puts("u r wrong\r\n\r");
83    main((unsigned __int64)"u r wrong\r\n\r", argv, v4);
84  }
85  else
86  {
87    puts("u r right!\r");
88  }
89  return 0;
```

```
89     return 0;
90 }
```

View Code

分析这段伪代码，可以得到关键部分如下：

- v22需为0，按'/'随手注释
- 如上图中所示，要使v22==0,所有v22=1的语句均不能运行，则需要：**v21==22**，69行判断均不进入，既要 **v15[i] != (*((_DWORD *)&v6 + i) - 1) / 2**，同时，**v16~v20依次等于48,56，50,51,125，即字符0823**｝（在相应数字上按r键把相应的ASCII码转换为字符）
  同样按'/'键随手注释
- 找到定义v15的代码处，则根据上述的关键条件，初步**猜测需要输入22位字符**，其中前17位存入字符串v15中，**后5位覆盖v16~v20的取值，使v16~v20分别等于0823**｝
- 继续向下分析，找到输入的代码块，经过分析可得到关键信息如上图，其中**v21即为输入的长度**，通过关键条件v21==22验证了猜测**输入字符串长度为22**，同时需要保证输入的**后五位为0823**｝

  向下分析v15需要满足的条件，着重分析**v15[i] != (*((_DWORD *)&v6 + i) - 1) / 2**，

  > &v6为取v6的地址；(_DWORD *)&v6强制转化为_DWORD型指针，即两个字节；((_DWORD *)&v6 + i)为从&v6向后取sizeof(_DWORD)*i个字节；*((_DWORD *)&v6 + i) - 1)为取从&v6向后取4i个字节的值
  > 关于地址与指针加减问题http://www.cnblogs.com/WangAoBo/p/6365114.html

- 如上，分析v6的赋值段代码，v6~v13为_int64型，占8个字节，int型为4个字节，化为16进制如上图
- v6~v14通过**小端存储方式**在内存中的存储情况如上

则可写出python脚本解得flag： *wctf{ElF_InX_Ckm_0823}*

python脚本如下：

```
v6=[0x0EF, 0x0C7, 0x0E9, 0x0CD, 0x0F7, 0x8B, 0x0D9,
    0x8D, 0x0BF, 0x0D9, 0x0DD, 0x0B1, 0x0BF, 0x87,
    0x0D7, 0x0DB, 0x0BF]

L=[]
for i in range(17):
    num = (v6[i] - 1)/2
    ans = chr(int(num))
    L.append(ans)


flag = ''.join(L)
flag+='0823}'

print(flag)
```

转载于:https://www.cnblogs.com/WangAoBo/p/6365987.html