

xctf BABYRE

原创

lcer. 于 2021-03-09 17:02:55 发布 59 收藏

分类专栏: [xctf-wp](#) 文章标签: [wp](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/weixin_43655690/article/details/114589698

版权



[xctf-wp](#) 专栏收录该内容

2 篇文章 0 订阅

订阅专栏

elf文件IDA打开后, 程序很简单

```
1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     char s; // [rsp+0h] [rbp-20h]
4     int v5; // [rsp+18h] [rbp-8h]
5     int i; // [rsp+1Ch] [rbp-4h]
6
7     for ( i = 0; i <= 181; ++i )
8     {
9         envp = (const char **)((unsigned __int8 *)judge + i) ^ 0xCu;
10        *((_BYTE *)judge + i) ^= 0xCu;
11    }
12    printf("Please input flag:", argv, envp);
13    __isoc99_scanf("%20s", &s);
14    v5 = strlen(&s);
15    if ( v5 == 14 && (unsigned int)judge((__int64)&s) )
16        puts("Right!");
17    else
18        puts("Wrong!");
19    return 0;
20 }
```

要求输入14个字符, 然后进入judge中进行相应操作, 那么judge是关键函数, 可是我们不能反编译这个函数, 可能是对函数进行了加密, 我们发现7-11行对judge进行了一系列操作, 应该是解密过程, 我们直接说解决的办法吧

这里应该有两种方式可以解决, 由于IDA python运用的还不是很熟练, 这里只说一种方式吧。

我们再15行这里设置断点（也就是在运行judge之前设置断点），这时judge已经解密，我们我们需要做的就是重新定义这里。具体如下

```
.data:0000000000600B00 public judge
.data:0000000000600B00 judge proc far ; CODE XREF: main+80fp
.data:0000000000600B00 ; DATA XREF: main+161r ...
.data:0000000000600B00 push rbp
.data:0000000000600B01 mov rbp, rsp
.data:0000000000600B04 mov [rbp-28h], rdi
.data:0000000000600B08 mov byte ptr [rbp-20h], 66h
.data:0000000000600B08 judge endp ; sp-analysis failed
```

在调试过程中（注意不能直接静态进行，因为一定要经过前面7-11行代码的解密），首先找到并选中judge这段，然后按d，将这段代码变为数据，然后按c在变为代码，然后右键选择create function(或者按p)，然后就可以f5反编译了

```
0 char v5; // [rsp+0h] [rbp-10h]
7 char v6; // [rsp+Ch] [rbp-1Ch]
8 char v7; // [rsp+Dh] [rbp-18h]
9 char v8; // [rsp+Ah] [rbp-1Ah]
10 char v9; // [rsp+Fh] [rbp-19h]
11 char v10; // [rsp+10h] [rbp-18h]
12 char v11; // [rsp+11h] [rbp-17h]
13 char v12; // [rsp+12h] [rbp-16h]
14 char v13; // [rsp+13h] [rbp-15h]
15 char v14; // [rsp+14h] [rbp-14h]
16 char v15; // [rsp+15h] [rbp-13h]
17 int i; // [rsp+24h] [rbp-4h]
18
19 v2 = 102;
20 v3 = 109;
21 v4 = 99;
22 v5 = 100;
23 v6 = 127;
24 v7 = 107;
25 v8 = 55;
26 v9 = 100;
27 v10 = 59;
28 v11 = 86;
29 v12 = 96;
30 v13 = 59;
31 v14 = 110;
32 v15 = 112;
33 for ( i = 0; i <= 13; ++i )
34 *(_BYTE *)(i + a1) ^= i;
35 for ( i = 0; i <= 13; ++i )
36 {
37     if ( *(_BYTE *)(i + a1) != *(&v2 + i) )
38         return 0LL;
39 }
40 return 1LL;
41 }
```

得到这个之后便可以写脚本了

```
v2 = [102, 109, 99, 100, 127, 107, 55, 100, 59, 86, 96, 59, 110, 112]
flag = ''
for i in range(len(v2)):
    flag += chr(v2[i]^i)
print(flag)
```

得到flag{n1c3_j0b}