xctf re2-cpp-is-awesome

原创

菜鸟m号 于 2019-10-01 22:11:18 发布 105 收藏

文章标签： xctf

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/jentle8/article/details/101851473

版权

这是一篇简单的wp，下载二进制文件在kali上面跑一下，可以看到是命令行函数估计是C++写的

```
root@kali:~/Desktop/xctfpwn# ./re2-cpp 20
Better luck next time
root@kali:~/Desktop/xctfpwn# ./re2-cpp 1381238273213628643
Better luck next time
root@kali:~/Desktop/xctfpwn#
```

然后file指令看一下文件属性：

```
root@kali:~/Desktop/xctfpwn# file re2-cpp
re2-cpp: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked
, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=0
8fba98083e7c1f7171fd17c82befdfe1dcbcc82, stripped
root@kali:~/Desktop/xctfpwn#
```

发现是64位x86的。

然后拖到ida分析：直接string进入到字符串找到我们需要的字符串位置，然后定位到函数所在位置：

```
LOAD:000…  0000000C  C  GLIBC_2.2.5
.rodata:…  00000078  C  L3t_ME_T3ll_YOu_SOm3thIng_ImpOrtant_A_{FL4G}_W0nt_b3_3X4ctly_…
.rodata:…  00000017  C  Better luck next time\n
.rodata:…  00000021  C  You should have the flag by now\n
.rodata:…  00000008  C  Usage:
.rodata:…  00000007  C   flag\n
```

发现有特殊字符串Better …我们定位到相关函数所在位置然后交叉引用一下，定位到主函数位置：

```
0000000000400B89
0000000000400B89                ; int __cdecl main(int, char **, char **)
0000000000400B89                main proc near
```

对伪C代码进行分析可以看到判断输入错误函数：

```
for ( i = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::begin(&v9); ; sub_400D7A
(&i) )
  {
    v11 = std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::end(&v9);
    if ( !sub_400D3D((__int64)&i, (__int64)&v11) )
      break;
    v6 = (_BYTE *)sub_400D9A((__int64)&i);
    if ( *v6 != off_6020A0[dword_6020C0[v12]] )
      sub_400B56();
    ++v12;
  }
```

可以锁定到那个输出luck函数的if语句是一个判断句，是按字符判断，v12=0一开始，doword_6020C0的位置是一串字符串：

```
00000000006020C0 dword_6020C0    dd 24h                  ; DATA XREF: main+DD↑r
00000000006020C4                 align 8
00000000006020C8                 db    5
00000000006020C9                 db    0
00000000006020CA                 db    0
00000000006020CB                 db    0
00000000006020CC                 db  36h ; 6
00000000006020CD                 db    0
00000000006020CE                 db    0
00000000006020CF                 db    0
00000000006020D0                 db  65h ; e
00000000006020D1                 db    0
00000000006020D2                 db    0
00000000006020D3                 db    0
00000000006020D4                 db    7
00000000006020D5                 db    0
00000000006020D6                 db    0
00000000006020D7                 db    0
00000000006020D8                 db  27h ; '
00000000006020D9                 db    0
00000000006020DA                 db    0
```

https://blog.csdn.net/jentle8

注意它是八字节一个字符！！！，然后再HEX窗口中打开，可以得到需要的字符串，
‘24’,‘0’,‘5’,‘36’,‘65’,‘7’,‘27’,‘26’,‘2d’,‘1’,‘3’,‘0’,‘d’,‘56’,‘1’,‘3’,‘65’,‘3’,‘2d’,‘16’,‘2’,‘15’,‘3’,‘65’,‘0’,‘29’,‘44’,‘44’,‘1’,‘44’,‘2b’。

off_6020A0保存了一堆字符：**L3t_ME_T3ll_Y0u_S0m3th1ng_1mp0rtant_A_{FL4G}_W0nt_b3_3X4ctly_th4t
**

那个明显就是6020C0保存的值作为字符串的某一个值，清楚了就开始写脚本：

```
x = "L3t_ME_T3ll_Y0u_S0m3th1ng_1mp0rtant_A_{FL4G}_W0nt_b3_3X4ctly_th4t_345y_t0_c4ptur3_H0wev3r_1T_w1ll_b3_C00l_1
F_Y0u_g0t_1t"
off = ['24','0','5','36','65','7','27','26','2d','1','3','0','d','56','1','3','65','3','2d','16','2','15','3','6
5','0','29','44','44','1','44','2b']

flag=""

for i in off:
    i=int(i,16)
    flag+=x[i]

print(flag)
```

运行得到flag: ALEXCTF{W3_L0v3_C_W1th_CL45535}