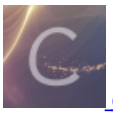
# XCTF-Reverse-ExerciseArea-006-writeup

原创

y4ung 于 2019-08-02 11:05:15 发布 4070 收藏

分类专栏： ctf 文章标签： ctf ReverseEngineering







## 0x00 介绍

本题是xctf攻防世界中Reverse的新手第六题。题目来源： `RC3 CTF 2016`

需要对该二进制文件logmein进行逆向分析，找到flag

实验环境：IDA Pro 7.0，gdb

## 0x01 解题过程

### 1.1 文件分析

1. 在Vscode中安装插件： `hexdump for VSCode` ，用Vscode打开，显示文件的十六进制：

可以看到文件的开头有 `ELF` ，说明这是一个在Linux下的可执行文件；

2. 在kali中用 file 命令，可以看到这是一个64bit的系统中编译的文件，因此你的Linux也必须是64bit才可以运行，同时可以看到该文件编译后符号表是被strip掉的

```
root@kali:~/hzy/ctf-learning# file logmein
logmein: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x8
6-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=c8f7fb137d9be24a19eb4f10efc29f7a421578a7, stripped
```

3. 修改文件权限为可执行，运行该文件。可以看到需要输入一个字符串密码，那么接下来要做的就是对文件逆向分析了

```
root@kali:~/hzy/ctf-learning# chmod +x logmein
root@kali:~/hzy/ctf-learning# ./logmein
Welcome to the RC3 secure password guesser.
To continue, you must enter the correct password.
Enter your guess: abcdef
Incorrect password!
```

## 1.2 脱壳

用IDA打开，发现该二进制文件未被加壳，因此不需要进行脱壳操作

## 1.3 逆向分析

1. 由于该二进制文件的符号表被strip了，因此用gdb调试，打断点的时候要用内存地址的形式来设置

由IDA可知，main函数起始地址为0x400630，gdb中在该地址处打断点：

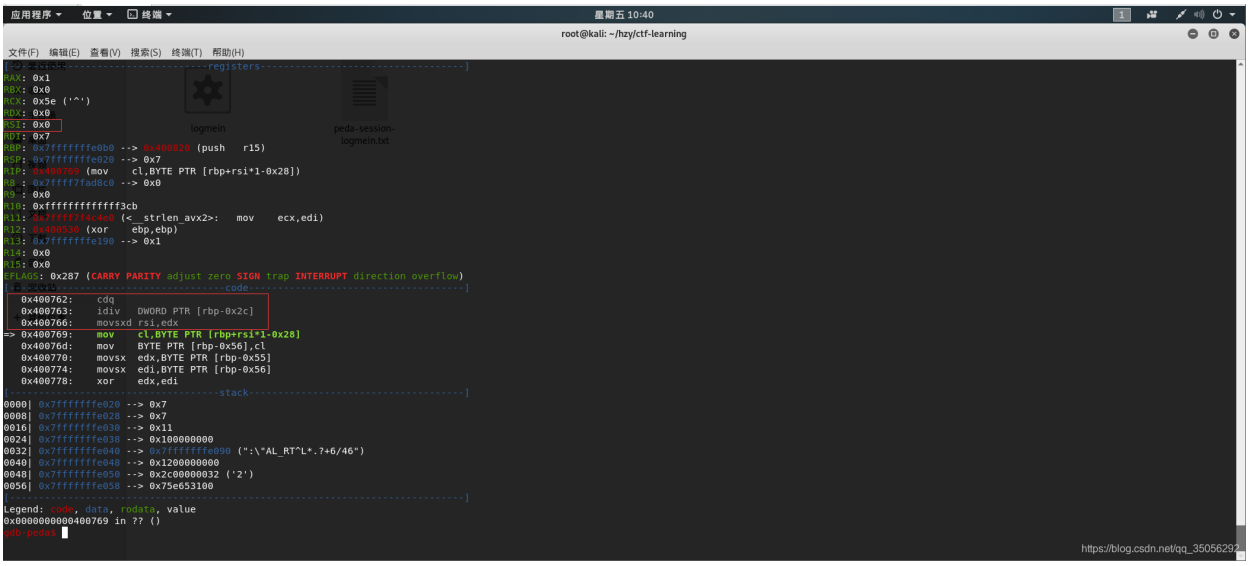
```
gdb-peda$ b *0x400630
Breakpoint 1 at 0x400630
```

运行过程中发现有两个字符串，rbp-0x20处为：`:"AL_RT^L*.?+6/46`，rbp-0x28处为：`harambe`；用户输入的字符串起始地址为 rbp-0x50

0x400630–0x4006F5：第一个基本块中，先获取用户输入字符串的长度，如果长度小于字符串 `:"AL_RT^L*.?+6/46`，则调用函数sub_4007C0，输入失败。用户输入字符串长度必须 ≥ 17

接下来是一个循环

1. rbp-0x54 保存的是循环循环变量
2. 0x400707–0x400725为循环退出条件：循环遍历的值 ≥ 用户输入字符串长度才跳转，并且调用函数sub_4007F0，提示输入正确
3. 在主要的循环体中：
    1. 0x40072B–0x400749保证循环变量的值＜字符串`:"AL_RT^L*.?+6/46`的长度
    2. 接下来是最主要的，0x400754–0x40078E中，将字符串`:"AL_RT^L*.?+6/46`和`harambe`对应位置的字符进行异或。其中，rbp-0x55存放的是字符串`:"AL_RT^L*.?+6/46`的字符，rbp-0x56存放的是`harambe`的字符，rbp-0x57存放的是对应字符异或得到的字符。再将异或后得到的字符与用户输入字符的对应位置进行比较，相同的话才不会调用函数sub_4007C0（输入失败），循环变量的值自增1。
    3. 那么就有一个疑问了，字符串`:"AL_RT^L*.?+6/46`和`harambe`的长度不相同怎么办？经过调试发现地址0x400762–0x400766的汇编码应该是让rsi寄存器存储着 rsi % len(`harambe`) 的值。这样一来，当循环变量的值超过`harambe`的长度时，就又回到0，从头开始取`harambe`的字符




可以用python脚本解出flag：

```
s1 = ':"AL_RT^L*.?+6/46'
s2 = 'harambeharambehar'

res = ""

for i in range(0, len(s2)):
    s1_a = ord(s1[i])
    s2_a = ord(s2[i])

    xor_res = s1_a ^ s2_a

    res = res + chr(xor_res)


print(res)
```

flag为：`RC3-2016-XORISGUD`