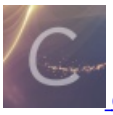
Dark CTF 2020-Rev/so_much-writeup

原创

y4ung 于 2020-10-07 09:24:08 发布 263 收藏 1

分类专栏： ctf 文章标签： ctf



1. 介绍

本题是dark ctf Reverse的第一题： `so_much` ，网址：https://ctf.darkarmy.xyz/challs
题目描述：strcmp printf

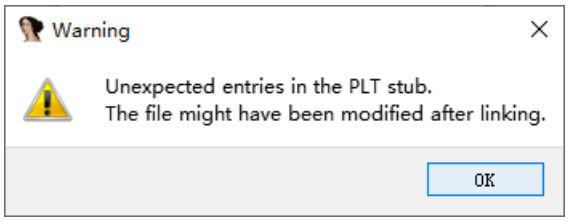
> 第一次打比赛，不过是自己一个，只写了Rev的题。8道Rev的题只做出来4道...还得继续努力

2. 分析

2.1 IDA静态分析

```
$ file so_much
so_much: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux
-x86-64.so.2, BuildID[sha1]=5a4b0c077367a5f40b197b32f9025ac3479fdcc6, for GNU/Linux 3.2.0, not stripped
$ chmod +x so_much
$ ./so_much
Let's have an argument. Shall we?
$ ./so_much 123
Nada. Not the right password!
*Psst the password is part of the flag*
```

扔进IDA里分析看看，发现报了好多警告：



先不管警告，进去看看再说。 `Let's have an argument. Shall we?` 在main函数中被使用。
main函数中，先将argv保存到v10，v10[1]就是用户的输入。接下来调用 `get_flag(v3, &v13, &v11, argv);` 函数，但是没有返回值。不过传进去的是变量的地址，所以应该还是可以修改变量的值的。
往下看，外层的if判断了参数的个数必须为2个，即./so_much 和 用户输入 这两个参数。然后将v11赋值给argv，但是后面没有用到argv了，所以这个操作好像没啥用。
然后在里层的if判断中，调用了sub_10C0函数进行判断： `if ( sub_10C0(v10[1], v11) )` ，只有当函数返回值为0时，才会打印出正确的flag。

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  __int64 v3; // rcx
  __int64 v4; // rbp
  const char *v5; // rdi
  __int64 v6; // rdx
  int result; // eax
  unsigned __int64 v8; // rcx
  unsigned __int64 v9; // rt1
  const char **v10; // [rsp-28h] [rbp-28h]
  const char **v11; // [rsp-18h] [rbp-18h]
  unsigned __int64 v12; // [rsp-10h] [rbp-10h]
  __int64 v13; // [rsp-8h] [rbp-8h]

  __asm { endbr64 }
  v13 = v4;
  v10 = argv;                                   // 先将argv保存到v10，v10[1]为用户的输入
  v12 = __readfsqword(0x28u);
  v11 = &unk_2008;
  get_flag(v3, &v13, &v11, argv);               // get_flag中对v10进行赋值
  if ( argc == 2 )
  {
    argv = v11;
    if ( sub_10C0(v10[1], v11) ) 函数返回值必须为0
    {
      v5 = "Nada. Not the right password!\n*Psst the password is part of the flag*";
    }
    else
    {
      argv = v11;
      sub_10B0();
      v5 = "WoW! so much revving...";
    }
    sub_1090();
  }
  else
  {
    v5 = "Let's have an argument. Shall we?";
    sub_1090();
  }
  result = 0;
  v9 = __readfsqword(0x28u);
  v8 = v9 ^ v12;
  if ( v9 != v12 )
    result = sub_10A0(v5, argv, v6, v8);
  return result;
}
```

以下为试错的过程，如果不想看的可以跳到2.2节

那么现在得先看一下get_flag函数的内容了。在get_flag函数里，a3是main函数中的v11，是我们需要关注的变量。但是a3只被v7赋值了。。。什么鬼。。而且for循环中的flag_48跟进去以后，一层套一层，从flag_48到了flag_125，把我看吐了= =||

```
unsigned __int64 __usercall get_flag@<rax>(__int64 a1@<rbp>, _QWORD *a2@<rdi>, __int64 a3@<rcx>, __int64 *a4@<rsi>)
{
  unsigned __int64 result; // rax
  unsigned __int64 v5; // rt1
  _QWORD *v6; // [rsp-70h] [rbp-70h]
  signed int i; // [rsp-5Ch] [rbp-5Ch]
  __int64 v8; // [rsp-58h] [rbp-58h]
  __int64 v9; // [rsp-50h] [rbp-50h]
  __int64 v10; // [rsp-48h] [rbp-48h]
  int v11; // [rsp-40h] [rbp-40h]
  __int16 v12; // [rsp-3Ch] [rbp-3Ch]
  __int64 v13; // [rsp-38h] [rbp-38h]
  __int64 v14; // [rsp-30h] [rbp-30h]
  __int64 v15; // [rsp-28h] [rbp-28h]
  __int64 v16; // [rsp-20h] [rbp-20h]
  __int64 v17; // [rsp-18h] [rbp-18h]
  unsigned __int64 v18; // [rsp-10h] [rbp-10h]
  __int64 v19; // [rsp-8h] [rbp-8h]

  __asm { endbr64 }
  v19 = a1;
  v6 = a2;
  v18 = __readfsqword(0x28u);
  v8 = 0LL;
  v9 = 0LL;
  v10 = 0LL;
  v11 = 0;
  v12 = 0;
  for ( i = 7; i <= 29; ++i )
  {
    v13 = 0LL;
    v14 = 0LL;
    v15 = 0LL;
    v16 = 0LL;
    v17 = 0LL;
    v13 = (unsigned __int8)flag_48((unsigned int)i);
    a4 = &v13;
    a2 = &v8;
    sub_10D0(&v8, &v13);
  }
  *v6 = &v8;
  v5 = __readfsqword(0x28u);
  result = v5 ^ v18;
  if ( v5 != v18 )
    result = sub_10A0(a2, a4, &v8, a3);
  return result;
}
```


而且有的函数本来是有参数的，我双击跟进去发现是类似下面的代码：

```
__int64 sub_10D0()
{
  __asm { endbr64 }
  return sub_1070();
}
```

再回到get_flag函数时，整个函数的参数就已经没了。总感觉IDA是不是F5出毛病了。。

```
// v3, &v13, &v11, argv
__int64 __usercall get_flag@<rax>(__int64 a1@<rcx>, __int64 a2@<rbp>, __int64 *a3@<rdi>, __int64 *a4@<rsi>)
{
  __int64 result; // rax
  unsigned __int64 v5; // rt1
  signed int i; // [rsp-5Ch] [rbp-5Ch]
  __int64 v7; // [rsp-58h] [rbp-58h]
  __int64 v8; // [rsp-50h] [rbp-50h]
  __int64 v9; // [rsp-48h] [rbp-48h]
  int v10; // [rsp-40h] [rbp-40h]
  __int16 v11; // [rsp-3Ch] [rbp-3Ch]
  __int64 v12; // [rsp-38h] [rbp-38h]
  __int64 v13; // [rsp-30h] [rbp-30h]
  __int64 v14; // [rsp-28h] [rbp-28h]
  __int64 v15; // [rsp-20h] [rbp-20h]
  __int64 v16; // [rsp-18h] [rbp-18h]
  unsigned __int64 v17; // [rsp-10h] [rbp-10h]
  __int64 v18; // [rsp-8h] [rbp-8h]

  __asm { endbr64 }
  v18 = a2;
  v17 = __readfsqword(0x28u);
  v7 = 0LL;
  v8 = 0LL;
  v9 = 0LL;
  v10 = 0;
  v11 = 0;
  for ( i = 7; i <= 29; ++i )                   // i∈[7,29]
  {
    v13 = 0LL;
    v14 = 0LL;
    v15 = 0LL;
    v16 = 0LL;
    v12 = flag_48(&v18, i);
    sub_10D0();
  }
  *a3 = &v7;
  v5 = __readfsqword(0x28u);
  result = v5 ^ v17;
  if ( v5 != v17 )
    result = sub_10A0();
  return result;
}
```


2.2 求助ghidra

索性扔到ghidra里看看呢。。一下子就看懂了，s2是内存地址0x2008开始的一个字符串，然后经过get_flag函数修改，在于用户的输入argv[1]比较，当strcmp返回值为0、即二者相等时，输出flag。

```
...
s2 = (char **)0x2008;
get_flag((int64_t)&s2);
if (argc == 2) {
    iVar1 = strcmp(argv[1], s2, s2);
    if (iVar1 == 0) {
        printf("darkCTF%s\n", s2);
        .plt.sec("WoW! so much revving...");
        ...
    ...
...
```

```
undefined8 main(uint32_t argc, char **argv)
{
    int32_t iVar1;
    undefined8 uVar2;
    int64_t in_FS_OFFSET;
    char **s1;
    uint32_t var_14h;
    char **s2;
    int64_t canary;

    canary = *(int64_t *)(in_FS_OFFSET + 0x28);
    // s2: ebp-0x10
    s2 = (char **)0x2008;
    get_flag((int64_t)&s2);
    if (argc == 2) {
        iVar1 = strcmp(argv[1], s2, s2);
        if (iVar1 == 0) {
            printf("darkCTF%s\n", s2);
            .plt.sec("WoW! so much revving...");
        } else {
            .plt.sec("Nada. Not the right password!\n*Psst the password is part of the flag*");
        }
    } else {
        .plt.sec("Let\'s have an argument. Shall we?");
    }
    uVar2 = 0;
    if (canary != *(int64_t *)(in_FS_OFFSET + 0x28)) {
        uVar2 = __stack_chk_fail();
    }
    return uVar2;
}
```


那就好办了，直接gdb动态调试，最后得到s2经过get_flag后的值为：`{w0w_s0_m4ny_funct10ns}`。

运行程序，得到flag：`darkCTF{w0w_s0_m4ny_funct10ns}`

```
$ ./so_much {w0w_s0_m4ny_funct10ns}

darkCTF{w0w_s0_m4ny_funct10ns}
WoW! so much revving...
```

3. 总结

有时候IDA识别不好的，可以试试ghidra，再不济，上gdb或ollydbg动态调试。

同时，题目中有时候会使用疑兵之计，比如本题中get_flag函数里for循环中的flag_xx函数系列。
如果当真搁那儿算半天，或是用python重写那几十个函数的调用，那可真是浪费时间了。
遇到这种工作量比较多的，不妨先往下看，把整个程序的脉络搞清楚，如果工作量大的那部分确确实实跳不过去，需要去逆，那再回头逆它也不迟。