三个白帽-来 PWN 我一下好吗 writeup

转载

weixin_34124577 于 2018-03-08 11:08:31 发布 99 收藏
文章标签： 数据结构与算法 shell
原文链接：https://juejin.im/post/5aa119af5188255569188ba9

> explorer · 2016/05/30 10:53

0x00 前言

三个白帽终于出了一个pwn的挑战。总算是能让我这种不会web的菜鸡拿几个猫币了。而且这一道题目出的很有心意，后面利用过程一环扣一环，质量很高。

现在就来分享一些详细的利用思路。

附上原文件：vms.zip

0x01 代码分析

做pwn题的第一步就是要分析代码，寻找漏洞。这次的程序代码流程不是很长，总共的函数数量也只有十几个，所以在分析代码方面不存在很多的问题。

结构体

这个是经过分析还原出来的Vulnerability结构体。也是程序中储存数据的主要结构。当然最重要的就是结构体里有一个指针。一旦覆盖可以用来任意读写

```
00000000 Vulnerability   struc ; (sizeof=0x60, mappedto_1)
00000000 rank            dq ?
00000008 titleSize       dq ?
00000010 detailLen       dq ?
00000018 title           db 64 dup(?)
00000058 detail          dq ?                    ; offset
00000060 Vulnerability   ends
复制代码
```

存在漏洞的

程序存在2个主要的漏洞 第一个是在edit修改Vulnerability结构的时候没有校验in_use位。所以存在即使内存已经被释放，却仍然能够修改内存的情况。也就是所谓的use after free漏洞

稍微分析一下，很容易就可以发现这一出的uaf漏洞

第二处漏洞就相对比较隐晦了。在add添加Vulnerability和edit修改Vulnerability结构的时候，如果输入的rank位负数，则程序会选择跳过而不是给rank字段赋值。我们可以利用这一点来leak堆上的信息。因为这是整个程序唯一一处不会破坏堆上原先数据的的的地方。Vulnerability结构的其他字段都会强制的修改内存中的数据，无法用来泄露信息

出题人挖的坑

出题人在整个题目里挖了3个大坑等着人来跳。要想拿到shell，需要一个一个的绕过。

1.所有堆指针加密

无论是存在.bss段的Vulnerability的指针还是Vulnerability结构中指向detail的指针，都是用一个随机数亦或之后再存放的。所以无法直接通过uaf来修改指针实现任意地址读写。

所以我们必须要先想办法搞到亦或用的随机数，否则无法完成利用。

2.堆指针的check函数

在程序通过指针读写detail内存之前都会有一个check函数来校验堆指针的有效性，如果校验错误就会直接退出程序

好在这个check并没有这个严格。一般来说只要保证Vulnerability的指针小于等于detail的指针的时候，有很大概率通过check。实际的测试下来，整个攻击一般在20次尝试以内就能成功

3.FULL RELRO保护

通过checksec就可以发现，程序开启了FULL RELRO，这意味着我们无法修改got表。即使通过uaf实现的任意读写，也无法控制eip。内存里也没有明显的函数指针能够被覆盖。

这里我通过覆盖free_hook来实现最后的调用system。这里是相关的代码。可以发现free函数可以通过hook来使他调用我们自定义的函数。只要我们修改__free_hook的值为system函数的地址，就可以拿到shell

```
2926    __libc_free (void *mem)
2927    {
2928      mstate ar_ptr;
2929      mchunkptr p;                          /* chunk corresponding to mem */
2930
2931      void (*hook) (void *, const void *)
2932        = atomic_forced_read (__free_hook);
2933      if (__builtin_expect (hook != NULL, 0))
2934        {
2935          (*hook)(mem, RETURN_ADDRESS (0));
2936          return;
2937        }
...
2962    }
2963    libc_hidden_def (__libc_free)
复制代码
```

0x02 详细利用流程

详细的利用代码我和我分析代码时的idb在这里
利用第一步就是要leak出用于亦或的随机数。需要用一些猥琐的思路来构建堆结构。

1.获得堆地址

首先，我们需要知道堆块的具体地址。这个我们可以leak堆管理中的链表指针来实现。Vulnerability结构的rank字段正好是存放链表指针的位置。正如前面所说的，只要我们在add的时候不写入rank的值就能够通过rank来leak堆的地址。

2.利用fastbin漏洞控制malloc内存地址

我们知道fastbin的单向链表链接的。而通过uaf，我们的rank字段正好可以覆盖这个单项链表指针。从而控制下一次malloc时候的地址。而且我们已经知道了堆的地址。所以我们就能控制malloc。使得新分配的Vulnerability结构的rank字段于另外一个Vulnerability的detail指针重合。这样我们就能够实现对detail字段的leak和修改了。而且结合刚才获得的堆地址。我们也可以反向算出用于亦或的随机数的数值。

到此，我们实现了任意地址的读写

3.泄露libc基地址

这个比较简单。因为normal bin是通过双向链表来管理的。而双向链表的头是存放在libc里的。所以只要在堆上随意的释放一块大的内存就能产生一个指向libc的指针。在我利用过程中。正好在堆上有这样一个指针，直接leak出来然后推算出libc基地址。

4.写入free_hook

用ida查看一下libc就能找到libc中free_hook的地址

最后的实际的地址是`0x3C0A10`。把算好的system函数的地址写入进去就可以了。最后找一块内存写入`/bin/sh`然后拿去free就可以完成整个利用的过程了

一血留念