

课程实验报告

课 程 名 称： 计算机系统实验

实验项目名称：

专 业 班 级：

姓 名：

学 号：

指 导 教 师：

完 成 时 间： 2018 年 5 月 22 日

信息科学与工程学院

|  |  |
| --- | --- |
| 实验题目：Buffer Lab | |
| 实验目的：This assignment will help you develop a detailed understanding of IA-32 calling conventions and stack organization. It involves applying a series of *buffer overflow attacks* on an executable file bufbomb in the lab directory. | |
| 实验环境：We generated the lab using gcc’s -m32 flag, so all code produced by the compiler follows IA-32 rules, even if the host is an x86-64 system. This should be enough to convince you that the compiler can use any calling convention it wants, so long as it’s consistent. | |
| 实验内容及操作步骤：        实验结果及分析：  屏幕快照%202018-05-21%20下午8  level0  getbuf中的test函数调用getbuf函数，接收字符串输入，然后返回到test函数。要求对getbuf进行缓冲区攻击，使getbuf函数不返回到test，而是调用smoke函数。      08049262 <getbuf>:  8049262: 55 push %ebp  8049263: 89 e5 mov %esp,%ebp  8049265: 83 ec 38 sub $0x38,%esp  8049268: 8d 45 d8 lea -0x28(%ebp),%eax  804926b: 89 04 24 mov %eax,(%esp)  804926e: e8 bf f9 ff ff call 8048c32 <Gets>  8049273: b8 01 00 00 00 mov $0x1,%eax  8049278: c9 leave  8049279: c3 ret  804927a: 90 nop  804927b: 90 nop  804927c: 90 nop  804927d: 90 nop  804927e: 90 nop  804927f: 90 nop   |  |  |  | | --- | --- | --- | | **地址** | **说明** | **寄存器** | | **Ebp+4** | Return address |  | | **ebp** | Old ebp | ebp | | **…** | … | … | | **Ebp-40** |  | eax | | **…** | … | … | | **Ebp-56** | eax | esp |     但是  当按照smoke的头地址写      显示有错误，问了同学发现：    改成0b或者1f都可以    level 1  它需要程序从getbuf()返回后执行fizz(int val)函数，并且为fizz(int val)函数传递你自己独有的cookie参数。      因为getbuf函数会将返回地址弹出并跳转到此地址（在这里也就是fizz的地址），fizz函数面对的栈帧就是ebp+8及其上面的部分，按照一般的函数调用规律来说，ebp+8处保存的是返回地址，从ebp+12处向上是输入参数。  所以，这次我们只需要从ebp-40的位置开始写44个字节，加上fizz的地址（从buf\_asm中可以得知是08048daf），加上4个字节（即fizz认为是返回值的部分），再加上val的值（也就是cookie，0x45875cb1）就可以了。   |  |  |  | | --- | --- | --- | | **地址** | **说明** | **指针** | | Ebp+12 | Fizz函数认为val所在的位置 |  | | Ebp+8 | Fizz函数认为返回值所在的位置 |  | | Ebp+4 | Return address |  | | ebp | Old ebp | Ebp | | Ebp-40 |  | Eax | | Ebp-56 |  | Esp | |  | Eax | Esp |     0x804d104这个地址存放的就是cookies的值；    level2  让bufbomb在getbuf()函数返回会执行bang()函数。但是在执行bang()函数之前我们需要设计全局变量global\_value为我们自己userid的cookie      要修改804d10c，使其变成cookie值  根据上面的反汇编代码第一行可以看到bang()函数的起始地址为0x08048d52    根据说明文档修改global\_value的值，并且3 ~ 4行将bang函数的起始地址压入栈，并返回，用于执行后跳转到bang函数。  运用如下gcc和objdump指令可以生成本机器的2进制代码  : gcc -m32 -c  : objdump -d    以上修改完成    为了执行这段代码，我们需要使getbuf返回到代码开头的地址，不妨就把这段代码放在缓冲区的开头部分，这样只需要跳转到eax即可。然后补全到44个字节，覆盖掉保存的ebp，再加上buffer的首地址就可以了。      因为步骤2中已经实现了修改值，并且跳转bang()函数的功能，所以我们需要将该代码放置地在buf中，并且让系统跳转到该段代码的起始处执行。  所以我们需要知道buf字符串在栈中开始地址。根据level0我们知道buf距离ebp 有0x28个字节。我们通过gbd调试获取执行到getbuf时的寄存器内，从而计算出buf的起始地址得到为0x55683558。  【步骤2中的代码序列（16字节）+填充序列（28字节）+填充跳转地址（4字节buf起始地址）】      buffer：55683558  level 3    这一次，除了需要执行我们放在栈上的代码之外，我们还需要改变程序的寄存器和内存状态并使程序在察觉不到的情况下正常返回。（也就是使攻击代码返回到调用getbuf的test，但是把返回值从1改成cookie。）    通过GDB得到保存的ebp值，防止在用字符串覆盖时破坏ebp的值  覆盖getbuf的返回值，使得getbuf返回到攻击代码的开头  在攻击代码中修改eax的值，并返回到正确的返回地址  第一步，得到ebp的地址  0x55683580    攻击代码所处的位置，也就是buffer开头的部分，与上一个任务相同，仍然是 0x55683558。阅读test函数可知，getbuf的返回地址应该是 0x8048cda          old ebp    556835b0      旧ebp  buffer  **level4**      （1）恢复SFP；  （2）设置getbufn返回值为cookie；  （3）跳转到testn中调用getbufn后的下一指令地址。    正常的程序运行过程中，具体在栈中的位置是不确定，所以该实验与level3的区别在于，buf等地址在栈中位置是变化的。这里需要用到空操作雪橇（nop sled）技术，通过nop指令构造序列，程序只要执行到任意一个nop指令就会逐渐执行到攻击代码      同样通过调试，或阅读testn函数的反汇编代码得到testn函数的ebp指针的内容为testn当前%esp+0x28。 //恢复ebp寄存器内容 getbufn()返回后执行的下一行指令地址获取方法与level3相同，地址为0x08048e3a      最大地址0x556833a8  将最高的buf地址作为跳转地址。        收获与体会：  这个实验让我弄懂了缓冲区溢出原理，以及堆栈的过程，函数调用的实现过程，函数传参的底层实现等问题。  从Level0开始，我明白了返回地址的位置，明白了是怎么样覆盖地址的。level1是修改参数，程序不会真正调用FIZZ，它只会执行它的代码。这很重要——在堆栈上放置cookie的位置的含义。  level2是修改全局变量，可以使用GDB来获得构造漏洞字符串所需的信息。在GETBUF中设置一个断点并运行到这个断点。确定参数，如全局地址的值和缓冲器的位置。手工确定指令序列的字节编码是繁琐的，容易出错。所以可以让工具通过编写包含要在堆栈上放置的指令和7个数据的汇编代码文件来完成所有的工作。用GCC—M32—C汇编这个文件，然后用objdump -d将其分解。于是能够得到在提示中键入的精确字节序列。  在编写汇编代码时还需要注意使用地址模式。在堆栈上推一个地址并使用RET指令。  level3是恢复栈结构，使程序执行您自己设计的机器代码。你这样做是一种隐秘的方式，程序没有意识到有什么不对劲。  level4是在level3第基础上实现随机化……  可以使用程序HE X 2 R A W来发送您的开发字符串的多个副本。如果在文件开发工具TXT中有一个副本，则可以使用以下命令：  unix> cat exploit.txt | ./hex2raw -n | ./bufbomb -n -u bovik  对于GETBUFN的所有5个执行，必须使用相同的字符串。否则，分级服务器使用的测试代码就会失败。  他的诀窍是利用NOP指令。它用一个字节编码（代码0x90）。课本第262页的“NOP SLID”。 | |
| 实  验成绩 |  |