|  |  |
| --- | --- |
| 实验题目：buflab | |
| 实验目的：这是关于内存溢出攻击的实验，相关内容仅限于学习，请不要将相关技术用于非法用途。 | |
| 实验环境：ubuntu12.04 | |
| 实验内容及操作步骤：  首先拿到cookie    正则表达式    打ctf时用Pwntools。这个实验里提供了hex2raw工具。  注意小端法      **Level0**    在test函数中调用getbuf,并且test检查栈，明显getbuf是溢出点。进入getbuf观察：    Buf地址是ebp-0x28，那么我们要多输入4个字节来覆盖ebp，才能到达ret address。  我们找一下smoke的地址    首地址为0x8048e0a  因为Gets的读取遇\0a就停止，所以字符串中间不能有\0a。 而smoke的首地址是0x08048e0a，用小端法，0a出现在中间，而我们要调用smoke之后就退出，我们不push ebp对后续使用没有影响，这样我们就可以直接跳过第一个指令，直接跳转到mov ebp, esp  写level0.txt      成功  **Level1**    就是要伪造fizz的参数。 32位系统是用栈传参，所以直接部署在栈上。而64位系统需要找布置参数的gadget。  我们知道，正常函数的栈帧结构都是：  Old ebp | ret address | arg1 | arg2 …  找到fizz函数地址：    地址为0x08048daf  因为fizz会exit退出，所以返回地址可以为任何值。Arg1为cookie.  写level1.txt      **Level2**  这一关很有意思，必须要用动态调试来获取栈地址，并在栈上布置shellcode。（没有开启aslr和NX）    这一关要求global\_value == cookie。 Global\_value在bss段，并且程序中没有gadget可以直接修改它。 根据题目描述，需要我们自己写shellcode。  思路：我们自己写.s文件，然后用objdump读出来操作码    首先获取cookie和global\_value的地址，然后编写shellcode  **注意：****AT&T风格的汇编，不加标注符的值，都是地址类型！ 立即数都会加$！！**  **所以是指把0x804d10c这个地方的值给eax。 因为0x804d10c是地址，所以自动寻址取值。 不需要再加（）来当成地址。**      .s文件是没有格式的！！ 这提供了一个很好的写shellcode的思路：用.s再as汇编，再用onjdump读。 就不用自己在网上找操作码了。  注意一个细节： b getbuf 断点会自动设置在准备完栈帧后！！ 并不是函数首地址！      得到buf首地址  可以看出，数值加上（），就是将这个数变成地址类型，并不代表对应内存空间的值。 取值的操作，只是由对应的汇编代码去执行的。  并且，gdb中，打印寄存器值，要在寄存器前加$,表示是值，而不是加%  不得不说，gdb的print命令真的很方便。 可以搭配汇编格式      成功  **Level3**  这一关要求跳回test，即call的下一条指令。 因为跳回时，不经过 pushl %ebp; movl %esp,%ebp 而之后会用ebp寻址，所以rop时必须恢复好test的ebp值。  正常情况下，我们rop一般跳到函数首地址，因为有movl %esp,%ebp，这样就不需要考虑ebp寻址的问题了。 而现在我们要跳到函数中间，会用到ebp寻址进行检测，所以要恢复ebp。  思路：类似level2, 写shellcode。 不同的是，这次还要动态调试，拿到old ebp。  方法一：在shellcode中movl $old ebp,%ebp  方法二：在getbuf的buf填写中，覆盖ebp为原值。  这里采用方法一。  在getbuf时，查询ebp处存的old ebp的值    要把数值当指针，需要进行类型转换！！  因为环境没开ASLR，所以buf首地址用level2的就行      拿到操作码    成功  **Level4**    即使不开ASLR，栈位置也有可能会变化。 原因之一:整个栈底会存argv和环境变量。 每次运行时，环境变量的不同会影响栈位置。  所以，用gdb查栈地址可能会与真实环境有出入。    **因为每次ebp不确定，所以我们在shellcode中采用 esp+偏移 的方式来取得ebp的值。**  Getbufn ret后，rsp就是testn的rsp，那么testn的ebp值就是rsp+0x28  **利用esp取ebp，可以通过lea指令**    下面我们要找buf的首地址。 因为栈位置不固定，所以我们要先了解到buf首地址的大致范围，从而好让其落在nop中。          可以发现，buf首地址每次差的并不是特别多。 我们有0x20c的空间，足以命中。  我们取buf地址为最高的，即0x55682fa8。 这样更容易命中在nop中。  写python脚本构建level4.txt      成功  实验结果及分析：  结合ida pro+gdb，成功通过所有关卡。  收获与体会：   1. 正则表达式还是要学的 2. 实践出真知，多用gdb。动态调试能力在程序复杂的时候非常重要。   3.对于一些字符串输入函数，都会以回车，空格作为结束符。 尽量把这些字节放在payload最后  4. **AT&T风格的汇编，不加标注符的值，都是地址类型！ 立即数都会加$！！**  **所以是指把0x804d10c这个地方的值给eax。 因为0x804d10c是地址，所以mov指令自动寻址取值。 不需要再加（）来当成地址。**    5..s文件是没有格式的！！ 这提供了一个很好的写shellcode的思路：用.s再as汇编，再用onjdump读。 就不用自己在网上找操作码了。  **因为每次ebp不确定，所以我们在shellcode中采用 esp+偏移 的方式来取得ebp的值。**  如：Getbufn ret后，rsp就是testn的rsp，那么testn的ebp值就是rsp+0x28  **利用esp取ebp，可以通过lea指令**  对于跳转，可以自己push再ret，和rop单独放置返回地址效果一样。    6. nop sled可以增加爆破效率 ，也可以用来够到栈上恰好够不到的位置上的数据当ret address  Vsyscall和nop等效 | |
| 实  验成绩 |  |