

ICS-Lab3说明文档

19302010020 袁逸聪

操作方式

终端输入 touch levelx.txt 创建txt文件用于输入答案

写完后, 输入 ./hex2raw <levelx.txt|.bufbomb -u 19302010020

-u 后为学生id, 用于生成cookie, 19302010020对应cookie为0x72e793cf

先用hex2raw把答案转化为二进制, 再作为输入运行bomb

level0

level0中, 系统将调用test, 再在test中调用getbuf

需要依靠没有溢出检测的getbuf修改返回地址, 调用smoke函数

1. disas smoke, 发现地址为0x08048bc6, 即需要把返回地址修改成这个, 下考虑返回地址的位置
2. 调用getbuf时, 用4字节保存返回地址, 而后进入getbuf
3. disas getbuf, 看到两次push占用8字节, 随后申请了0x24即36空间, 故在储存返回地址后, 栈中又存放了44字节
4. 由此, 任意填充44字节后, 在接下来的4字节输入0x08048bc6, 考虑到小端法, 应输入c6 8b 04 08

输入 ./hex2raw <level0.txt|.bufbomb -u 19302010020 检验通过

level1

level1与level0相似, 但仅仅调用函数不够, 还要传入正确参数

不调用smoke而调用fizz, 并且传入等同于cookie的参数

1. disas fizz, 发现地址为0x08048c01, 并且cmp对象为edx、eax, edx中数据来自于ebp+8, 显然是参数保存位置
2. 类似于level10, getbug的返回地址被更改为fizz的开头
3. push ebp, 并使ebp指向了esp所在, 即保存ebp的地址(也是fizz地址被防止的地址)
4. 此后, ebp未被更改, 因而ebp+8指向的是fizz地址所在地址之上8位
5. 由此, 任意填充44字节后, 输入0x08048c01(保存在读取参数时ebp位置), 空开4字节再输入cookie: 0x72e793cf(保存在读取参数时ebp+8位置)

输入 ./hex2raw <level1.txt|.bufbomb -u 19302010020 检验通过

level2

类似于level1, 调用bang, 但不是要参数, 而是要全局变量与cookie相等

因而需要先找到全局变量的位置, 自己写汇编代码修改全局变量并跳转到bang, 修改地址执行我们写的代码内容

1. `disas bang` 寻找全局变量位置，直接找到唯一的 `cmp`，比较 `eax` 与 `edx`，可见其中一者保存了 `cookie`，一者为需要修改的全局变量。顺便记录 `bang` 地址 `0x08048c63`
2. 但其地址是基于 `%ebx` 计算获得的，`run test` 时找到 `ebx` 被设为 `0x0804d000`，分别按照 `bang` 加上 `0x1114, 0x111c` 后 `print`，发现 `0x0804e114` 中存储正是 `cookie`，那么 `0x0804e11c` 就是应修改的全局变量了
3. 由此，需要做的事是：push `bang` 地址，修改全局变量为 `cookie`，`return`(调用 `bang`)

编写汇编代码 `assem2.s`: `push $0x08048c63 movl $0x72e793cf,0x0804e11c ret`

终端输入 `gcc -m32 -c assem2.s objdump -d assem2.o > assem2.d cat assem2.d`

看到对应的二进制代码 `68 63 8c 04 08 c7 05 1c e1 04 08 cf 93 e7 72 c3`

于是，将以上内容填充金 `level2` 答案的前44个字节部分，并把最后的返回地址覆盖设置为我们输入的汇编代码对应地址

1. 在 `getbuf` 中找到输入内容将被存储的缓冲区地址，即 `lea -0x28(%ebp)` 所赋值，设置断点查看接受赋值的 `edx`，找到为 `0x55683b18`
2. 由此，在前44个字节中，先填充上述二进制代码，随意补完44字节后，再输入 `0x55683b18`

输入 `./hex2raw <level2.txt|./bufbomb -u 19302010020` 检验通过

level3

`level3` 要求我们在执行攻击后继续继续正常运行 `test`，但需要把 `getbuf` 的返回值改为 `cookie`

类似于更改全局变量，更改寄存器 `eax` 需要我们自己写汇编代码执行，但为使 `test` 顺利运行，还需要恢复 `ebp` 与 `ebx` 的值(`getbuf` 函数结尾原本就会做)

1. `disas test` 找到调用 `getbuf` 后要执行代码的地址 `0x08048cf1`
2. 在调用 `getbuf` 处设置断点，获取需要保存的 `ebp:0x55683b60` 和 `ebx:0x0804d000`，用于恢复

编写汇编代码 `assem3.s`: `push $0x08048cf1 movl $0x72e793cf,%eax movl $0x55683b60,%ebp movl $0x0804d000,%ebx ret`

终端输入 `gcc -m32 -c assem3.s objdump -d assem3.o > assem3.d cat assem3.d`

看到对应的二进制代码 `68 f1 8c 04 08 b8 cf 93 e7 72 bd 60 3b 68 55 bb 00 d0 04 08 c3`

类似于 `level2`，将汇编代码的部分替换成以上内容(当然，更长的部分也需要替换更多的 `00`，以保证总长度不变)

输入 `./hex2raw <level3.txt|./bufbomb -u 19302010020` 检验通过

level4

`level4` 采用了栈地址随机化(栈建立在全局变量之上，根据需储存内容的不同，栈地址有所变化)，本关无法精确掌握需跳转地址

同时，`getbuf` 也变为 `getbufn`，申请 `0x204` 而不是 `0x24` 个空间用于写入

本关的随机化方式为在栈前申请随机大小的空间，因而栈指针 `esp` 的具体值无法实现确定，`ebp` 同理

- 需要提供5个输入，改用 `./hex2raw -n <level4.txt|./bufbomb -n -u 19302010020`

本关需求类似于level3，需要在修改返回值为cookie的同时恢复ebp、ebx并返回test，此处的ebp无法实现确定，只能根据当时的esp获取

1. `disas getbufn`，调用它时将保存返回地址、`push ebp`、`push ebx`、申请0x204空间，故书写时到第525~528个字节为返回地址
2. 在调用`getbufn`处设置断点，获取需要恢复的`ebx`：0804d000
3. 需要恢复的`ebp`无法直接获取，但可以在注入代码中根据`esp`反推：从`testn`中同步开始算，`push ebx`时`esp-4`，开辟空间又减少0x14，故`ebp`应为`esp+0x18`
4. `disas testn`，获取需要继续运行的下一行地址0x08048d81

编写汇编代码`assem4.s`: `movl $0x72e793cf,%eax lea 0x18(%esp),%ebp movl $0x0804d000,%ebx push $0x08048d81 ret`

这里push需要放在恢复ebp后，否则esp就被改变了

终端输入 `gcc -m32 -c assem4.s objdump -d assem4.o > assem4.d cat assem4.d`

看到对应的二进制代码 `b8 cf 93 e7 72 8d ac 24 08 02 00 00 bb 00 d0 04 08 68 81 8d 04 08 c3`

我们需要把这段代码写入栈中，并把原来的`return adress`替换成代码的首字节地址，但由于栈的具体地址随机化，无法精确确定

运行发现，level4的栈随机化并非完全随机，只是5次运行会改变顺序取几个值，断点检测每一次`getbufn`时写入内容的首地址：

0x55683998是其中最大的

也就是说，跳转到这个地址，可以保证接下来执行的是我们写入的区域而不是之前的其他代码

但由于随机，不一定正好落在首地址上(如果首地址小于0x55683998，就将落在中段)

因而采用`nop sled`策略，在前面部分大量填充`nop`(0x90)，效果为执行下一条，落入写入范围的情况下，必然能一路跳`nop`达到注入代码块

由此：

答案包含528个字节，后4个为0x55683998，紧跟在前面为上述汇编代码，其他位置用`nop`填充

输入 `cat level4.txt|./hex2raw -n|./bufbomb -n -u 19302010020` 检验通过