**Lab 3：拆弹实验**

姓名：李玉轩 学号：2023212248 班级：23DX0220 指导教师：杜海文、孟文龙 日期：2025.6.1

|  |  |
| --- | --- |
| **实验内容** | CMU Bomb Lab系列实验：  使用 GDB、objdump 等工具，阅读汇编代码并找出每个关卡的通关字符串（密码），避免引爆任何炸弹，最终实现程序的正常退出 |
| Phase1分析代码 0000000000400ee0 <phase\_1>:   400ee0: 48 83 ec 08 sub $0x8,%rsp #栈指针减法，留出空间   400ee4: be 00 24 40 00 mov $0x402400,%esi #esi存调用函数第一个参数  400ee9: e8 4a 04 00 00 callq 401338 <strings\_not\_equal> #比较函数   400eee: 85 c0 test %eax,%eax #eax存返回值   400ef0: 74 05 je 400ef7 <phase\_1+0x17>#根据返回值判断跳转到bomb   400ef2: e8 43 05 00 00 callq 40143a <explode\_bomb># bomb   * 所以比较函数前面的那一步存储的就是要比较字符串  操作 fig: 结果 Border relations with Canada have never been better.  fig: Phase2分析代码 400efe: 48 83 ec 28 sub $0x28,%rsp#栈指针移动   400f02: 48 89 e6 mov %rsp,%rsi#栈顶值存储到rsi   400f05: e8 52 05 00 00 callq 40145c <read\_six\_numbers>#读取六个数字   400f0a: 83 3c 24 01 cmpl $0x1,(%rsp)#比较栈上第一个数字是否为 1   400f0e: 74 20 je 400f30 <phase\_2+0x34>#相等跳转   400f10: e8 25 05 00 00 callq 40143a <explode\_bomb>#不相等爆炸   400f15: eb 19 jmp 400f30 <phase\_2+0x34>#跳过后续检查   400f17: 8b 43 fc mov -0x4(%rbx),%eax# 赋值   400f1a: 01 c0 add %eax,%eax#加倍   400f1c: 39 03 cmp %eax,(%rbx)#加倍之后跟输入是否相等   400f1e: 74 05 je 400f25 <phase\_2+0x29># 跳转   400f20: e8 15 05 00 00 callq 40143a <explode\_bomb>#不相等爆炸   400f25: 48 83 c3 04 add $0x4,%rbx#rbx中地址加4   400f29: 48 39 eb cmp %rbp,%rbx#检查是否已检查完所有数字   400f2c: 75 e9 jne 400f17 <phase\_2+0x1b>#如果没有检查完，继续循环   400f2e: eb 0c jmp 400f3c <phase\_2+0x40>#跳转到结尾   * 所以是找六个数字，并且第一个是1 * 然后一直循环，直到 %rbp,%rbx相等，也就是一个一个数字的读取 * 每个数字的比较：eax一直在加倍然后跟(%rbx)比较，所以栈中每一元素是它上一元素的两倍  结果 1 2 4 8 16 32  fig: Phase3分析代码 400f5b: e8 90 fc ff ff callq 400bf0 <\_\_isoc99\_sscanf@plt>#调用 sscanf 函数，从用户输入读取数据并存储到栈上。  400f60: 83 f8 01 cmp $0x1,%eax#比较 sscanf 的返回值是否等于1  400f63: 7f 05 jg 400f6a <phase\_3+0x27>#如果等于1，跳转  400f65: e8 d0 04 00 00 callq 40143a <explode\_bomb>#如果不等于1bomb   400f6a: 83 7c 24 08 07 cmpl $0x7,0x8(%rsp)#栈上第一个数字是否大于 7  400f6f: 77 3c ja 400fad <phase\_3+0x6a>#大于bomb  400f71: 8b 44 24 08 mov 0x8(%rsp),%eax#将栈上第一个数字加载到 %eax  400f75: ff 24 c5 70 24 40 00 jmpq \*0x402470(,%rax,8)#跳转  400f91: b8 85 01 00 00 mov $0x185,%eax    400f96: eb 26 jmp 400fbe <phase\_3+0x7b>   400fbe: 3b 44 24 0c cmp 0xc(%rsp),%eax#比较 %eax 与栈上第二个数字  400fc2: 74 05 je 400fc9 <phase\_3+0x86>#相等跳转  400fc4: e8 71 04 00 00 callq 40143a <explode\_bomb># 不相等bomb 实操和思路  * 第一个不爆炸是比较 sscanf 的返回值是否等于1，要想等于1，就要正确输入且个数正确   + 作为被调用的函数，参数在mov $0x4025cf,%esi，所以查内存   fig:   * + 发现要输入数字，而且是两个 * 第二个不爆炸   + cmpl $0x7,0x8(%rsp)栈上第一个数字是否大于 7，大于爆炸 * 第三个   + mov 0x8(%rsp),%eax#将栈上第一个数字加载到 %eax jmpq \*0x402470(,%rax,8)#跳转 这两部说明下面的数字跟第一个填入的数字有关   + 然后对照switch-case跳转表，输3，跳到0x402470 + (3 \* 8) = 0x402470 + 0x18 = 0x402488，看他里面存放的地址   fig:   * + 对应 $0x100 ，十进制256q  结果 3 256  fig: Phase4分析代码 000000000040100c <phase\_4>:  401024: e8 c7 fb ff ff callq 400bf0 <\_\_isoc99\_sscanf@plt>#调用scanf  401029: 83 f8 02 cmp $0x2,%eax#比较返回值是不是2  40102c: 75 07 jne 401035 <phase\_4+0x29>#bomb  40102e: 83 7c 24 08 0e cmpl $0xe,0x8(%rsp)#比较第一个值是否小于14  401033: 76 05 jbe 40103a <phase\_4+0x2e>  401035: e8 00 04 00 00 callq 40143a <explode\_bomb>  40103a: ba 0e 00 00 00 mov $0xe,%edx#赋值14  40103f: be 00 00 00 00 mov $0x0,%esi  401044: 8b 7c 24 08 mov 0x8(%rsp),%edi#将第一个解析的值加载到 %edi  401048: e8 81 ff ff ff callq 400fce <func4>#调用  40104d: 85 c0 test %eax,%eax# 返回值不是0bomb  40104f: 75 07 jne 401058 <phase\_4+0x4c>  401051: 83 7c 24 0c 00 cmpl $0x0,0xc(%rsp)#比较第二个值是否为零，就是第二个必修是0  0000000000400fce <func4>:  400fd8: c1 e9 1f shr $0x1f,%ecx# %ecx逻辑右移31位，补0，取最高位  400fdb: 01 c8 add %ecx,%eax# 拿自己的最高位加上result；（负数加1正数加0）14+0  400fdd: d1 f8 sar %eax # 算术右移，单操作数是只移动一位 7  400fdf: 8d 0c 30 lea (%rax,%rsi,1),%ecx  400fe2: 39 f9 cmp %edi,%ecx# 比较第一个输入值和%ecx的关系    400ff2: b8 00 00 00 00 mov $0x0,%eax# 给出0  400ff7: 39 f9 cmp %edi,%ecx # 再比较  400ff9: 7d 0c jge 401007 <func4+0x39> # 大于等于 function4具体分析 这段代码实现了一个递归函数 func4，其行为类似于一个二分搜索算法。具体步骤如下：   1. 计算中间值 %ecx。 2. 比较中间值 %ecx 和目标值 %edi。 3. 如果 %edi 小于或等于 %ecx，递归调用 func4，调整上限。 4. 如果 %edi 大于 %ecx，递归调用 func4，调整下限。 5. 返回结果  分析  * 要先保证scanf输入正确fig: * 401051: 83 7c 24 0c 00 cmpl $0x0,0xc(%rsp)#比较第二个值是否为零，就是第二个必修是0，所以第一个是7  结果 7 0  fig: Phase5分析代码 0000000000401062 <phase\_5>:  401067: 48 89 fb mov %rdi,%rbx#将 %rdi的值（函数的第一个参数）存储到 %rbx  40106a: 64 48 8b 04 25 28 00 mov %fs:0x28,%rax#从线程局部存储（TLS）中读取栈保护变量的值到 %rax  401071: 00 00   401073: 48 89 44 24 18 mov %rax,0x18(%rsp)#栈保护变量的值存储到栈空间中  401078: 31 c0 xor %eax,%eax#清零  40107a: e8 9c 02 00 00 callq 40131b <string\_length>#调用   40107f: 83 f8 06 cmp $0x6,%eax#字符串长度是否等于 6  401082: 74 4e je 4010d2 <phase\_5+0x70>  401084: e8 b1 03 00 00 callq 40143a <explode\_bomb>#不等于bomb  401089: eb 47 jmp 4010d2 <phase\_5+0x70>  40108b: 0f b6 0c 03 movzbl (%rbx,%rax,1),%ecx#从输入字符串中读取一个字节到 %ecx  40108f: 88 0c 24 mov %cl,(%rsp)#将 %ecx 的值存储到栈空间中   401099: 0f b6 92 b0 24 40 00 movzbl 0x4024b0(%rdx),%edx#从地址 0x4024b0 加上 %rdx 的值处读取一个字节到 %edx。  4010a0: 88 54 04 10 mov %dl,0x10(%rsp,%rax,1)  4010a4: 48 83 c0 01 add $0x1,%rax#将 %rax 的值加 1    4010bd: e8 76 02 00 00 callq 401338 <strings\_not\_equal>#调用    4010d2: b8 00 00 00 00 mov $0x0,%eax#栈保护   * 输入的字符作为偏移量，可以刚好使得0x4024b0作为首地址偏移所对应得新的字符与0x40245e对应得字符依次相等   fig:   * 所以偏移量 9 15 14 5 6 7前面使用了0xf作为掩码，所以可以使用字符串“9?>567”代替低ascll码得输入  结果 9?>567  fig: Phase6分析代码 00000000004010f4 <phase\_6>: #第一部分，最终有用部分，输入六个数字  401106: e8 51 03 00 00 callq 40145c <read\_six\_numbers>#需要输入6个数字 # 嵌套循环，外层循环作用是，确定输入这几个数字均在1-6之间，内层循环作用是，确定这几个数字互不相等  401117: 41 8b 45 00 mov 0x0(%r13),%eax  40111b: 83 e8 01 sub $0x1,%eax#数字的值-1  40111e: 83 f8 05 cmp $0x5,%eax#减完以后和5进行比较  401121: 76 05 jbe 401128 <phase\_6+0x34>#低于或者相等  401123: e8 12 03 00 00 callq 40143a <explode\_bomb>#否则爆炸  401130: 74 21 je 401153 <phase\_6+0x5f>#跳出外层循环  401132: 44 89 e3 mov %r12d,%ebx#将这个会变化的值（第一次是1）赋值给一个被调用者寄存器  40113e: 75 05 jne 401145 <phase\_6+0x51>#不等于才是对的  401140: e8 f5 02 00 00 callq 40143a <explode\_bomb>#相等爆炸 #第二部分，将输入的数字分别转换为7-该数字，栈中的每一个元素之间的地址间隔8个字节  401153: 48 8d 74 24 18 lea 0x18(%rsp),%rsi    40116d: 75 f1 jne 401160 <phase\_6+0x6c>  #第三部分，栈中最后会按照原来对应的数字来保存不同的地址   40116f: be 00 00 00 00 mov $0x0,%esi   4011a9: eb cb jmp 401176 <phase\_6+0x82> #第四部分，把栈中所有的下一个地址元素赋值给，其上一个地址元素+8对应的内存中  4011ab: 48 8b 5c 24 20 mov 0x20(%rsp),%rbx   4011d0: eb eb jmp 4011bd <phase\_6+0xc9> #第五部分，使用一个循环将栈中所有的地址元素对应的值，和它的下一个地址元素对应的值进行比较  4011d2: 48 c7 42 08 00 00 00 movq $0x0,0x8(%rdx)    401203: c3 retq  原 转换为7-该数字 栈内最终被覆盖成 6 1 0x6032d0 5 2 0x6032e0 4 3 0x6032f0 3 4 0x603300 2 5 0x603310 1 6 0x603320   * 查看地址   fig:   * 也就是按node的从小到大排序4 3 2 1 6 5  结果 4 3 2 1 6 5  fig: secret  1. **输入格式**：    * mov $0x402619,%esi：这条指令将输入格式设置为 "%d %d %s"。    * mov $0x603870,%edi：这条指令将输入的字符串存储到地址 0x603870。 2. **触发隐藏关卡的输入**：    * 程序会从 0x603870 读取输入的字符串，并尝试匹配格式 "%d %d %s"。  secret\_phase  1. **读取输入并转换为数字**：    * callq 0x40149e <read\_line>：读取用户输入的字符串。    * callq 0x400bd0 <strtol@plt>：将输入的字符串转换为十进制数字。    * 转换后的数字保存在 %rbx 中。 2. **数字范围检查**：    * lea -0x1(%rax),%eax：将转换后的数字减去 1。    * cmp $0x3e8,%eax：将减去 1 后的数字与 0x3e8（十进制 1000）进行比较。    * 如果减去 1 后的数字大于 0x3e8，则调用 explode\_bomb，炸弹爆炸。  大概思路fun7  * 如果输入的数字 %esi 大于当前节点的值 %edx，则递归调用 fun7，并将 %rdi 设置为当前节点的右子节点（0x8(%rdi)）。 * 如果输入的数字 %esi 小于当前节点的值 %edx，则递归调用 fun7，并将 %rdi 设置为当前节点的左子节点（0x10(%rdi)）。fig:  结果 fig: | |