**深圳大学实验报告**

**课程名称： 计算机系统(2)**

**实验项目名称： 逆向工程实验**

**学院： 计算机与软件学院**

**专业： 计算机科学与技术**

**指导教师： 马晨琳**

**报告人： 郑雨婷 学号： 2021150122 班级： 高性能**

**实验时间： 2023年4月13日～ 2023年4月24日**

**实验报告提交时间： 2023年4月24日**

**教务处制**

|  |
| --- |
| **一、实验目的：**   1. 理解程序（控制语句、函数、返回值、堆栈结构）是如何运行的 2. 掌握GDB调试工具和objdump反汇编工具 |
| **二、实验环境：7**   1. 计算机（Intel CPU） 2. Linux64位操作系统 3. GDB调试工具 4. objdump反汇编工具 |
| **三、实验内容**  本实验设计为一个黑客拆解二进制炸弹的游戏。我们仅给黑客（同学）提供一个二进制可执行文件bomb和主函数所在的源程序bomb.c，不提供每个关卡的源代码。程序运行中有6个关卡（6个phase），每个关卡需要用户输入正确的字符串或数字才能通关，否则会引爆炸弹（打印出一条错误信息，并导致评分下降）！  要求同学运用**GDB调试工具和objdump反汇编工具**，通过分析汇编代码**，**找到在每个phase程序段中，引导程序跳转到“explode\_bomb”程序段的地方，并分析其成功跳转的条件，以此为突破口寻找应该在命令行输入何种字符串来通关。  本实验要求解决Phase\_1、Phase\_2、Phase\_3、Phase\_4、Phase\_5、Phase\_6。通过截图把结果写在实验报告上。 |
| **四、实验步骤和结果**  **1.输入反汇编命令查看汇编代码（保存在1.txt文件中）**  ***$ objdump -d bomb\_64.txt > 1.txt***    将事先编译好的二进制文件进行反汇编，并将汇编代码重保存到1.txt文件中.  **2.对反汇编出来的代码进行分析，并完成六个关卡。**  (1)phase 1:  ①获取汇编代码：    ②分析代码逻辑：  340行将栈指针减8，开栈。  341行将地址0x401af8中的内容复制到寄存器%esi中。  342行调用函数string\_not\_equal，该函数将寄存器%esi中存放的内容作为参数，判断字符串与输入是否相同，相同则返回0，不相同则返回1；  343、344两行表示当%eax为0时跳转到400e78,否则跳转到explode\_bomb函数。  因此，phase1只需要输入与地址0x401af8中的内容相同的字符串即可跳过炸弹。所以使用gdb调试，获取0x401af的字符串。    分析得出要输入的字符串为：Science isn’t about why, it’s about why not?  ③进行测试：’    第一次运行时出现了以上错误，没有操作权限，只需要输入chmod +x bomb\_64.txt即可正常运行。输入；Science isn’t about why, it’s about why not?    答案正确，第一关通过！  (2)phase2  ①获取汇编代码：    ②分析代码逻辑：    350-353是将数据保存到栈中，372-376是将数据还原。  在phase2的主体中，首先调用函数read\_six\_numbers读取了六个数，这六个数存在rsp指针以上的位置。在循环中%rbp每次加4，可以猜测是int类型（占4个字节的数据类型）。    第一个爆炸条件为M[rbp]!=M[rbp+12]，也就是数字与其之后的第三个数字不等；      跳出循环的标志是%rbp=%r13,%r13在进入循环之前设置成了%rsp+12。因为每次%rbp+4,三次循环后就会跳出。这样不引爆炸弹的条件为前三个数等于后三个数。  第二个引爆炸弹的条件是在跳出循环之后，%r12d不等于0；%r12d是在循环体之内数字的累加，也就是说6个数不能累加为0。    综上所述，第二关输入的六个数应该满足前三个数等于后三个数，并且六个数的累加和不为0.  ③进行测试：  输入1 2 3 1 2 3    答案正确，第二关通过！  (3)phase3  ①获取汇编代码：    根据rax不同得到不同的地址  ②分析代码逻辑：  在381-384行，准备了前四个参数，第三个参数存在%rdx中，第四个参数存在%rcx中，占四个字节，为int类型。第二个参数%esi利用gdb查看后如下：    说明这一关输入两个int型的数据，分别存储在地址为%rsp+12和%rsp+8的内存中，当输入不对时引发爆炸。  调用函数输入数后，返回为大于1即可避开第一个引爆条件。  若M[rsp+12]>7,爆炸；因此第一个输入应在0~7之间。  将第一个输入存入%eax后第 92行进行跳转。目的地址是 M[0x401b60 + rax\*8]，其中%rax是输入的第一个整数。当跳转到对应的地址后，将%eax赋对应值，现在我们只需找出输入0~7后分别会跳转到那些地址。利用gdb查看：    再与phase3中的代码对应得到：   |  |  |  | | --- | --- | --- | | 第一个参数 | 跳转地址 | eax值 | | 0 | 0x400f32 | 0x217=535 | | 1 | 0x400f6f | 0x39e=926 | | 2 | 0x400f39 | 0xd6=214 | | 3 | 0x400f40 | 0x153=339 | | 4 | 0x400f47 | 0x77=119 | | 5 | 0x400f4e | 0x160=352 | | 6 | 0x400f55 | 0x397=919 | | 7 | 0x400f5c | 0x19c=412 |   当第二个参数等于%eax时，拆弹成功。  ③进行测试：  输入：2 214    答案正确，第三关通过！  (4)phase4  ①获取汇编代码：    ②分析代码逻辑：  与第三关相同，第三个参数放在%rdx中，第二个参数放在%esi中，利用gdb查看%esi中的数据：    说明本关需要输入一个int数，当输入不对时引发爆炸。。  若输入返回值不为1当场爆炸。  若输入大于0，将输入存入%edi，进入函数func4;  下面分析func4函数，传入参数%edi，先保存数据，最后还原数据，在函数主题部分首先判断参数%esi是否小于等于1，若是，则跳出，跳到还原数据。若不小于1，将参数减1继续作为新的参数存入%edi，再次进入函数func4，得到的返回值存入%ebp. 将参数减2继续作为新的参数存入%edi，再次进入函数func4，得到的返回值与减1得到的返回值相加。这实现了求斐波那契数列的第n项。    在主函数中，若斐波那契数列函数返回值为55,则不会爆炸。通过数学计算可知，斐波那契数列第九项为55.因此，我们要输入9.  ③进行测试：    答案正确，第四关通过！  (5)phase 5  ①获取汇编代码：    ②分析代码逻辑：  与第三关相同，需要输入两个整数，输入不对时引发爆炸。    保留输入第一个数的低4位.若等于1111则引爆炸弹。    每次%edx加1,当%eax等于15时才跳出循环,每一次将%eax累加到%ecx。    跳出循环后，将%edx(循环次数)与12比较，若循环次数不为12，触发爆炸。因此我们需要找到一个初始%rax使得循环12次时%eax的值正好为15。最后第二个参数为满足这种情况时的sum值。  通过gdb调试出401065处的数组：    用最简单直接的方法——穷举法。寻找满足条件的初始下标，得到结果为7时，12次跳转后eax的值为15，累加和为93。    ③进行测试：  输入：7 93    答案正确，第五关通过！  (6)phase 6  ①获取汇编代码：    ②分析代码逻辑：  程序首先调用函数strtol@plt，将字符串转换为long型数据存储在地址为0x602780的内存中，通过gdb调试将0x602780的内存值打印出来，是一张表。    程序是将输入的数插入了这张表的表头，调用函数func6进行两层循环降序排序，并且通过三次加8偏移寻址更新%rax使之为表里的第四个数值的地址，然后比较地址为%rax和地址为0x602780的值是否相同，即排序后的第四个数值和输入的数值进行比较，如果不同则引爆炸弹。由此可知，程序输入一个数值，并将它插入原有的数据表后进行降序排序，输入的数值必须排在第四个位置才不会引爆炸弹，也就是输入的数值必须介于600和673之间。  ③进行测试：  输入：618    答案正确，第六关通过！  **3.最终结果**    结果正确！拆除炸弹成功！ |
| **三、实验总结与体会：**  本次实验是希望我们通过分析汇编语言，来推测出跳转到引爆函数的条件，从而避开引爆条，让我巩固了mov、call、j、test、cmp、add等汇编指令的用法，了解程序（控制语句、函数、返回值、堆栈结构）是如何运行的，并且还掌握了GDB调试工具中p这个打印功能以及objdump反汇编工具的使用。  我认为这是一次十分困难的实验，第一关参照示例还能较为轻松的写出，但第二关开始就难度升高，尽管将每一行的指令操作都翻译出来也不一定能看出程序的功能，需要理解栈的结构，还需要知道数据类型所占的字节，涵盖了许多方面的知识。 |

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字：马晨琳  2023年4月 日 |
| 备注： |