**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： 信息与计算科学（数学与计算机实验班）**

**指导教师： 罗胜**

**报告人： 王曦 学号： 2021192010 班级： 数计班**

**实验时间： 2023年04月30日至04月30日**

**实验报告提交时间： 2023年04月30日**

**教务处制**

|  |
| --- |
| **一、 实验目标与要求：**   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：**   1. 计算机（Intel CPU） 2. Linux64位操作系统（Ubuntu 17） 3. GDB调试工具 4. objdump反汇编工具 |
| **三、实验方法与步骤：**  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\_64和主函数所在的源程序bomb\_64.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。  本实验需解决Phase\_1(**15分**)、Phase\_2(**15分**)、Phase\_3(**15分**)、Phase\_4(**15分**)、Phase\_5(**15分**)、Phase\_6(**10分**)。通过**截图+文字**的形式把实验过程写在实验报告上，最后并撰写**实验结论与心得(15分**)。 |
| **四、实验过程及内容：**  0. 准备工作  (1) 将”bomb\_64”和”bomb\_64.c”发送到实验所用的用户的home目录下的exp3文件夹中.  (2) 因实验所用的用户非root用户, 可能对"bomb\_64”无执行权限, 故用” sudo chmod +x bomb\_64”为文件所有者添加执行权限, 用”ls -la bomb\_64”查看用户对”bomb\_64”的权限, 发现用户有”-x”权限, 如下图所示:    图0.1: 为当前用户添加对”bomb\_64”的执行权限  (3) 新建一个”1.txt”文件, 用于存储后续得到的”bomb\_64”的汇编代码. 因实验用户在当前文件夹下无创建文件的权限, 故先用”sudo touch 1.txt”命令创建文件, 再用”sudo chown wangxi\_2021192010 1.txt”将文件所有者从root改为实验用户. 用”sudo chmod +w 1.txt”命令为文件所有者添加写权限, 再用”ls -la 1.txt”查看权限, 如下图所示:    图0.2: 新建”1.txt”文件并设置权限  (4) 用”sudo objdump -d bomb\_64 > 1.txt”命令对”bomb\_64”反汇编, 将汇编代码输出到”1.txt”中, 如下图所示:    图0.3: 将”bomb\_64”反汇编   1. 为方便查看汇编代码, 用Final Shell的文件资源管理器打开”1.txt”, 如下图所示.     图0.4: 查看”bomb\_64”的汇编代码  **1. 第一关**  (1) 定位到<phase\_1>处, 阅读汇编代码, 其功能如下图注释所示:    图1.1: <phase\_1>的汇编代码   1. 发现程序调用了string\_not\_equal()函数, 猜测其功能为判断两字符串是否相等. 调用该函数前的mov指令为该函数准备参数, 猜测内存地址为0x401af8的变量即用于比较的字符串. 2. 用”gdb ./bomb\_64”命令使用gdb调试”bomb\_64”, 如下图所示:     图1.2: 用gdb调试”bomb\_64”   1. 用”p (char\*)0x401af8”命令查看地址0x401af8处的字符串, 发现第一关的答案为 ”Science isn't about why, it's about why not?”, 如下图所示:     图1.3: 查看地址0x401af8处的字符串   1. 验证第一关的答案, 发现答案正确, 如下图所示:     图1.4: 验证第一关的答案  **2. 第二关**  (1) 定位到<phase\_2>处, 阅读汇编代码, 其功能如下图注释所示:    图2.1: <phase\_2>的汇编代码   1. 程序先调用read\_six\_numbers()函数, 猜测其功能为从内存中读取6个元素, 不妨设为a[0], ..., a[5]. 初始时r13 = rsp + 12, rbp = rsp. 进入循环, 循环终止条件为r13 = rbp, 而每次循环令rbp += 4, 则循环3 次. 循环时, 每次判断M[rbp + 12]与M[rbp]是否相等, 若不相等则炸弹爆炸; 否则令r12 = a[i], 故a[i] = a[i + 3]. 循环结束后, 若r12寄存器中的值为0, 则炸弹爆炸. 2. 综上, 6个元素满足a[0] = a[3], a[1] = a[4], a[2] = a[5], 且a[0] + a[1] + a[2] != 0. 取序列a = [1, 1, 1, 1, 1, 1], 验证第二关的答案, 发现答案正确, 如下图所示:     图2.2: 验证第二关的答案  **3. 第三关**  (1) 定位到<phase\_3>处, 阅读汇编代码, 其功能如下图注释所示:    图3.1: <phase\_3>的汇编代码   1. 用gdb查看地址0x401ebe处的字符串, 猜测本行的代码为”scanf(“%d %d”, 寄存器1, 寄存器2)”.     图3.2: 查看地址0x401ebe处的字符串   1. 程序的逻辑为输入两个整数, 若第一个数 > 7, 则炸弹爆炸, 进而第一个数 <= 7. 若未爆炸, 根据第一个数为0 ~ 7分别跳转, 其中第一个数为7时炸弹爆炸. 分析知有如下7组合法解(十进制输入):   ① 0, 535; ②2, 214; ③ 3, 339; ④ 4, 119; ⑤ 5, 352; ⑥ 6, 919.  (4) 以解 ① 为例, 验证第三关的答案, 发现答案正确, 如下图所示:    图3.3: 验证第三关的答案  **4. 第四关**  (1) 定位到<phase\_4>处, 阅读汇编代码, 其功能如下图注释所示:    图4.1: <phase\_4>的汇编代码   1. 用gdb查看地址0x401ec1处的字符串, 猜测本行的代码为”scanf(“%d”, 寄存器)”.     图4.2: 查看地址0x401ec1处的字符串   1. 程序读入一个整数, 若该整数 < 1, 则炸弹爆炸; 否则调用递归函数func4(). 对该函数, 递归终止条件为参数 <= 1时返回0, 否则返回func4(n - 1) +func4(n - 2), 即求Fibonacci数列   的第n项, 并判断其是否为55. 问题转化为: Fibonacci数列的第几项(下标从1开始)为55. 因Fibonacci数列的前几项为1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ..., 故55为Fibonacci数列的第9项, 即第四关的答案为9.   1. 验证第四关的答案, 发现答案正确, 如下图所示:     图4.3: 验证第四关的答案  **5. 第五关**  (1) 定位到<phase\_5>处, 阅读汇编代码, 其功能如下图注释所示:    图5.1: <phase\_5>的汇编代码   1. 用gdb查看地址0x401ebe处的字符串, 猜测本行的代码为”scanf(“%d % d”, 寄存器1, 寄存器2)”.     图5.2: 查看地址0x401ebe处的字符串   1. 程序读入两个int, 若第一个int为15则炸弹爆炸; 否则执行如下的伪代码:  |  | | --- | | int a[12] = { ? };  int k, ans;  scanf(%d % d”, &k, &ans);  int sum = 0;  for (int i = 0; i < 12; i++) {  sum += a[k];  k = a[k];  }  check(sum == ans); |   其功能为判断序列中元素的累加是否为输入的第二个int, 若不是则炸弹爆炸.   1. 因循环终止条件为eax = 15, 而eax = M[0x401ba0 + rax \* 4], 则该内存处可能是一个长度为16的数组. 在gdb中用”p \*0x401ba0@16”查看数组中的16个元素, 如下图所示:     图5.3: 查看地址0x401ba0处的数组   1. 根据k的值倒推得每次循环对应的k值和a[k]值如下:  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | i | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | | k | 6 | 14 | 2 | 1 | 10 | 0 | 8 | 4 | 9 | 13 | 11 | 7 | | a[k] | 15 | 6 | 14 | 2 | 1 | 10 | 0 | 8 | 4 | 9 | 13 | 11 |   故k的初始值为7, a[]中元素的累加和为93, 即答案为”7 93”.   1. 验证第五关的答案, 发现答案正确, 如下图所示:     图5.4: 验证第五关的答案  **6. 第六关**  (1) 定位到<phase\_6>处, 阅读汇编代码, 其功能如下图注释所示:      图6.1: <phase\_6>的汇编代码   1. 程序调用strtol()函数, 将字符串转化为long型, 存储在地址0x602780处, 并将其置于edi寄存器中作为参数调用fun6()函数. 2. 用于判断炸弹是否爆炸的寄存器的值在401108指令处被修改, 用gdb在地址0x401108处设置断点, 如下图所示:     图6.2: 在地址0x401108处设置断点   1. 运行程序直至断点处, 用”x/wx $rax”查看此时rax寄存器中的值, 发现为0x258, 即十进制的600, 故答案为600:     图6.3: 在断点处查看rax寄存器中的值   1. 验证第六关的答案, 发现答案正确, 如下图所示:     图6.4: 验证第六关的答案 |
| **五、实验结论：**  成功拆除6个关卡的炸弹, 并发现有一个隐藏关卡, 但未知触发条件. 可能可通过Ollydbg等支持修改的反汇编工具修改跳转地址以触发隐藏关卡.  各关卡答案:   1. 第一关: Science isn't about why, it's about why not? 2. 第二关: 1 1 1 1 1 1 (答案不唯一) 3. 第三关: 0 535 (答案不唯一) 4. 第四关: 9 5. 第五关: 7 93 6. 第六关: 600 |
| **六、心得体会：**  (1) 初始时对x86的汇编指令不熟悉, 阅读较为困难. 随着实验的进行, 逐渐熟悉了x86的汇编指令, 实验进行速度明显加快.  (2) 阅读有l、q、s等后缀的指令时, 可先忽略后缀, 分析整体的功能, 需要具体到二进制数位时才需带上后缀分析. 本实验中, 大部分情况下可忽略后缀.  (3) 根据程序运行的复杂程度, 选择合适的跟踪寄存器的值的方式. 对较为简单的顺序、条件、循环程序, 用纸笔模拟寄存器的值的变化较为方便; 对涉及到复杂的栈帧操作、递归、多级寻址等的程序, 用gdb设置断点, 在程序运行到断点处时打印寄存器的值较为方便. 此外, 一个取巧但效率极高的方法是: 先整体阅读汇编程序, 将其转化为C语言代码, 运行时加入中间变量的输出.  (4) 编写调用函数的汇编程序时, 需注意使用的寄存器是调用者保存还是被调用者保存, 注意调用前保存函数中会修改的寄存器的值, 并在返回前复原对应的值. 若函数有多个return且使用被调用者保存的寄存器时, 注意每个return前都需恢复寄存器的值.  (5) 熟悉常用数据类型的大小和常用的十六进制有利于计算偏移地址.  (6) 在Linux下实验时需注意文件的权限, 即使是当前用户创建的文件, 也未必有写和执行的权限, 这一特性虽使得用户操作较为繁琐, 但有利于保证系统的安全, 如不允许用户执行自己目录下的恶意代码. |

|  |
| --- |
| 指导教师批阅意见：  成绩评定：  指导教师签字：  年 月 日 |
| 备注： |