**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（创新班）**

**指导教师： 刘 刚**

**报告人： 何泽锋 学号： 2022150221 班级： 高性能特色班**

**实验时间： 2024年4月26日～ 2024年5月16日**

**实验报告提交时间： 2024年5月16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：**   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**  **Phase\_1：**  ①查看汇编代码发现，此处将0x401af8的地址传给了%esi，然后调用了<strings\_not\_equal>函数，根据函数名判断可知，这是一个用与判断字符串是否相等的函数，而刚刚传入%esi的地址可能是作为参数传入该函数，因此此处查看该地址开始的字符串    ②使用指令查看可知，此处存的是一段英文，因此需要判断输入的英文与该句子是否相同    总结：这是一段if语句，通过判断输入的字符串是否与默认字符串相等，运行结果如下    **Phase\_2:**  ①任务二完整汇编代码：    ②首先将当前寄存器的数据压栈，保存好指针的位置，可以看到此处%rsp下移0x48个位置，存放的数据占用了0x20个，剩余空间用于存放后方调用<read\_six\_numbers>读入的数据    ③调用输入函数，并将栈首地址传给%rbp，此处%rbp担任循环的控制变量    ④这两段代码主要是进行初始化工作，分析后续代码可知%r13存放的是循环跳出的地址，，%r12d是用于求和的寄存器    ⑤此处的两个寄存器%eax和%rbx，分别存储的是%rsp和%rsp+0xc的数据，根据字节分析得知存储的是四字节即int数据，比较%eax和%rbx中的数据    ⑥根据上一比较结果（两数是否相等），若相等则将该数大小加入%r12d中，将%rbp往后移int数据大小，比较下一对数据，若到达循环边界则跳出。    ⑦将%r12的的数据与上自身，若%r12d是0则结果为0，其余结果不为0，因此此处代表数据的和不能为0    ⑧恢复原始寄存器，结束函数调用      ⑨总结：这是一个循环，输入六个数，要求每隔三个数相等，可以理解为两个一样的数据组，并且数据组的和要不为0，以下举例可行解：   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 数据1 | 数据2 | 数据3 | 数据4 | 数据5 | 数据6 | | 1 | 2 | 3 | 1 | 2 | 3 |   ⑩结果展示：    **Phase\_3：**  ①完整代码展示      ②观察代码可以看到，此处将0x401ebe地址传给了%esi，查看该地址的内容可知，意思是输入两个int数据    ③判断输入的第一个数是否小于等于7，若是满足则将该数放入%eax，结合后续代码可以分析得出，这是一个switch语句    ④根据输入的第一个数的值寻找跳转位置，此处乘8是因为地址是8个字节    ⑤检查从0x400000开始的连续8个8字节内存单元，可以看到对应的跳转地址    ⑥跳转到对应位置，会将输入的位置与当前位置的一个值比较，具体对应关系如下所示：       |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 第一个数 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | | %eax（H） | 0x217 | 0x39e | 0xd6 | 0x153 | 0x77 | 0x160 | 0x397 | 0x19c | | %eax（D） | 535 | 926 | 214 | 339 | 119 | 352 | 919 | 412 |   ⑦将输入的第二个数与对应位置的数比较，若相等则恢复栈指针，并返回    ⑧结果举例：    **Phase\_4:**  ①完整代码展示：    ②阅读代码可以看到将0x401ec1放到了%esi中，因此查看该地址所存放的内容，可以看到此处显示%d，因此推测输入数据为一个int型数据    ③调用输入函数，若输入成功会返回1，否则不是1并跳转到bomb指令    ④分析可知输入的数据存放在%rsp+0xc的地址内，此处判断输入是否大于0    ⑤将输入的数据作为参数传入<func4>，将函数返回值与0x47（55）比较是否相等    ⑥分析<func4>，函数每次调用都会保存%rbx、%rbp的原始数据，将传入参数赋给%ebx，将%eax赋值为1，比较传入的参数是否大于等于1，若是，则将%edi-1，同时递归调用<func4>，当递归到最深层时%edi=1，此时满足跳出条件，将%eax作为返回值返回。%ebp会将返回的值累加，在每一次递归，当递归完%edi-1时，会递归%edi-2，同时将前一递归结果相加。    ⑦将<func4>按照逻辑写成c++，运行后可以得到结果    ⑧可以看到，当edi输入的值为9时，其返回值会是55。进一步了解可以知道，这是求解斐波那契数列的第9项，因此输入的值为9即可通过本题。    ⑨答案输入展示：    **Phase\_5:**  ①完整代码展示：      ②查看0x401ebe内容，可以发现需要输入两个int型数据    ③根据输入值的返回值，确定返回值是否小于等于1，若不是则bomb    ④分析可知%rsp+0xc是第一个输入的数，and 0xf意思是取该输入十六进制的最后一位因此输入数据实际范围是0-15    ⑤初始化操作，结合后续代码分析可知，%ecx用于累加数据，并最终与%rsp+0x8即第二个输入数据比较是否相等，%edx是记录操作次数，结束时需要操作12次，否则bomb。cltq将32位寄存器扩展位64位。    ⑥分析可知从0x401ba0开始取数据，%eax数据范围是0-15，四个字节为一位，可以推断出此处存了16个int数据    ⑦从此处开始查看16个int型数据    ⑧结合两个个判断条件可以倒推出输入数据：（1）%eax=15 （2）%edx=12    ⑨根据数据存储可以做出表格   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 数据下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | 数据值 | 10 | 2 | 14 | 7 | 8 | 12 | 15 | 11 | 0 | 4 | 1 | 13 | 3 | 9 | 6 | 5 | | 选取顺序 | 7 | 9 | 10 |  | 5 |  | 12 | 1 | 6 | 4 | 8 | 2 |  | 3 | 11 |  |   按照选取顺序重新排序   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 数据下标 | 7 | 11 | 13 | 9 | 4 | 8 | 0 | 10 | 1 | 2 | 14 | 6 | 3 | 5 | 12 | 15 | | 数据值 | 11 | 13 | 9 | 4 | 8 | 0 | 10 | 1 | 2 | 14 | 6 | 15 | 7 | 12 | 3 | 5 | | 选取顺序 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |  |  |  |  |   因此可以知道输入的第一个数据为7，第二个数据为  11+13+9+4+8+0+10+1+2+14+6+15=93  ⑩结果验证：    **Phase\_6:**  ①完整代码展示：    ②分析可知此处是将输入的数据进行一定处理后放到%edi中，作为参数传入<fun6>，因此需要结合<fun6>分析    ③<fun6>完整代码展示：      ④观察代码可以发现多次使用了地址0x602780，因此查看此附近空间可得到如图表，进一步分析可以知道这是一个链表，首位是数据，第二位为下标，第三位下一个数据所在地址    ⑤输出运行<fun6>之后的表，可以知道将这10个数从大到小排序，最后会放回第四大的数据，忽略输入的值将其他数据排序可以得到如图所示表，因此当输入的数大于673，则返回673，小于673大于600会返回输入的数，小于600会返回600   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 826 | 782 | 673 | 600 | 529 | 488 | 374 | 370 | 286 |     ⑥输入小于600会放回600    ⑦输入小于673大于600会返回输入的数    ⑧输入大于673，则返回673    **隐藏关：**  (1)观察文档可以看到，在phase\_6下方还有一个<secretr\_phase>，直接运行程序无法直接到达此关，因此搜索文档寻找调用<secretr\_phase>的地方    (2)可以看到在<phase\_defused>函数调用了<secretr\_phase>    (3)0x602c80存储的是通过的题目数量，只用当前6题都对了的情况下才能进入隐藏关      (4)需要输入两个数一个是int，另一个是string    (5)可以看到，此处显示了9，回看前面题目输入，发现这是phase\_4的答案，因此入口可能与phase\_4有关    (6)根据返回值判断输入的是否为两个数与上方“%d %s”相对应    (7)此处调用了一个判断函数，传入了两个参数，检查此参数，可以发现这应该是另一个需要输入的数据，结合前面的题目可知此地址是第四题输入，因此可知隐藏关的进入条件是在第四题原来输入9的基础上再输入 austinpowers      (8)进入隐藏关后可以看到此处读取了输入数据并将输入数据存到%rax，运行并查看寄存器后可知，此处存储的地址为0x603120    (9)再次检查寄存器可以知道，<strtil@plt>是将输入的字符串转为int型，然后让%eax=输入数据-1，与1000进行比较，实际意义为输入数据需要小于等于1000    (10)将输入的数据和地址0x6025a0作为参数传入函数    (11)<fun7>完整代码展示    (12)查看0x6025a0附件地址空间，可以看到按照8字节存放，数据与跳转地址相间存储    (13)根据上方数据和地址可以分析出该数据是一棵平衡二叉树，转换为图形表示如下：    (14)将<fun7>写成c++代码如下所示，<secret\_phase>最后比较%eax和0x3可知，<fun7>的返回值需要为3，根据递归关系可知，递归最后一层返回0，上一层为2\*0+1=1，再往上为2\*1+1=3，因此根据其返回可知初始输入结果为107    (15)输入107验证答案，可以看到答案正确。    **全部题目解答正确，所有关卡通过！** |
| **五、实验总结：**  本次实验成功通过了6道题和隐藏关，较为清晰的理解到gdb的使用，包括如何通过gdb查看汇编代码、寄存器信息、地址信息等。同时通过这几道题目，对各个类型的语句的汇编表示也都更加清晰。也学习了几种数据结构汇编表示，包括链表，二叉树。最为重要的是，查阅这些汇编代码后，对汇编代码也跟更加理解，能够更快的理解到前后代码之间的关系。 |

|  |
| --- |
| **指导教师批阅意见：**  **成绩评定：**  指导教师签字：刘刚  2024年5月 日 |
| 备注： |