|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 课程编号 1502760001-07  题目类型 实验3 | |  |  |  | | --- | --- | --- | | **得分** | **教师签名** | **批改日期** | |  | 冯禹洪 |  | |

**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： 软件工程（腾班）**

**指导教师： 冯禹洪**

**报告人： 黄亮铭 学号： 2022155028 班级： 腾班**

**实验时间： 2024 年 4 月 27 日至 5 月 10 日**

**实验报告提交时间： 2024 年 5 月 10 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：**  前期准备   1. 使用命令为提供读写权限。 2. 使用命令 对bomb文件进行反汇编，并将结果输出到1.txt中。 3. 使用打开文件，并定位到main函数中。 4. 阅读main函数，然后跳转到相应的关卡。   第一关  阅读*phase\_1*的代码。我发现程序首先申请栈空间，然后将0x401af8处的内容存储到寄存集*%esi*中。接下来调用函数*strings\_not\_equal*，合理猜测这个函数的作用是判断字符串和系统给定的字符串是否相等，相等返回0，否则返回其他。因为*%eai*是存储函数返回值的寄存器，0x400e80处判断返回值是否为0，如果为0则跳转。综上所述，我们需要输入一个字符串，使其和0x401af8处的内容相等。    我们进入GDB调试获取0x401af8处的内容。    发现是上图所示的内容，所以在测试阶段输入上述内容即可通过当前关卡。  第二关  阅读*phase\_2*的代码。我发现程序（0x400e8c-0x400ea0）首先申请栈空间将需要用到的寄存器的值保存起来并在函数结束时还原（0x400ee0-0x400ef4）。接下来将栈指针赋给*%rsi*，作为参数传入*read\_six\_numbers*。根据函数名可知，该函数的作用是读取6个数字，这6个数字需要符合一定规则。  继续阅读代码。*read\_six\_numbers*函数调用结束后，程序将*%rsp*将赋值给*%rsi*作为迭代指针。然后将*%rsp+12*处的值赋给*%r13*作为循环结束的判断条件，清空*%r12*将其作为累加器。  接下来程序进入循环（0x400eba-0x400ed4），将%rbp赋值给%rbx，将%rbp+12指向的内存地址的内容赋值给%eax，然后比较%eax的数据和%rbp指向的内存地址的数据是否相等。如果不相等则触发炸弹，如果相等则跳过触发炸弹的函数。再将%rbx指向的内存地址的数据累加到%r12。最后迭代指针加4即指向下一个数据，然后判断%r13和%rbp是否相等即是否遍历到第4个数据，如是则退出循坏，否则继续循环。  然后判断累加器%r12d是否等于0，如果等于0，则触发炸弹，否则不触发。最后执行上面提到的还原操作。    通过上述分析我们可以合理猜测：程序需要我们输入6个数字，这6个数需要满足第1和第4、第2和第5、第3和第6个数字相等，并且前3个数字相加的和不能为0。  为了印证我的猜测，阅读*read\_six\_numbers*，发现额外规则：如果输入数字个数小于6也会触发炸弹。相应的字符串存储在0x401eb2。    使用GDB调试查看内存0x401eb2处的内容。    发现是上图所示的内容，我们只需要输入满足规则的数字即可过关。  第三关  阅读phase\_3代码。程序首先开辟栈空间，将存储数据的地址存储到寄存器中。然后将0x401ebe的赋值给%esi，合理猜测这是我们需要输入的内容（后续可以使用GDB调试查看需要我们输入的内容是什么）。接下来是读入数据，代码保证数据有两个，否则会触发炸弹。再判断第一个输入的数据是否大于7，大于7则跳转并触发炸弹，否则将第一个输入的数据搬移到寄存器%eax中，根据公式\*0x401b60+8\*%eax跳转到内存相应地址处。最后将相应的数据搬移到%eax中，并与第二个输入的数据比较大小，如果不相等则触发炸弹，反之则过关。    启动GDB调试。通过输入x发现\*（int\*）（0x401b60+%rax\*x）跳转的地址如下图所示。我们只需要输入的y和输入的x跳转的地址后被赋值的%eax相等即可。该关卡一共有8组合法解：0｜535, 1｜926, 2｜214, 3｜339, 4｜119, 5｜352, 6｜919, 7｜412。    使用GDB调试查看0x401ebe处的内容，发现确实是需要我们输入两个数字。    第四关  阅读phase\_4的代码。程序首先申请栈空间，将指针赋值给%rdx，将输入内容的地址存储到%esi中，将累加器（计算输入数据的个数）清空。然后调用scanf函数读取数据，数据内容存放到%rdx存储的地址中。接着判断输入数据的个数，如果不等于1则触发炸弹。再比较输入数据和0的大小，小于或等于0则触发炸弹，否则跳转调用函数func4，在调用函数前将输入数据搬移到寄存器%edi中，作为函数的参数。在函数调用结束后，比较返回值和0x37的大小，不相等则触发炸弹，反之过关。最后释放申请的空间。    阅读func4的代码。首先申请空间保存即将用到的寄存器中的内容。然后将%edi中的数据搬移到%ebx中，将%eax赋值为1，作为函数无法递归到下一层时的返回值。比较参数%edi和1的大小，小于或等于1时跳转到函数恢复寄存器内容，释放空间的阶段，否则执行两个递归函数（分别将%edi-1和%edi-2作为新参数）。在执行完第一个递归函数后，将返回值临时存储到%ebp中。在执行完第二个递归函数后，将%ebp中的数据加到%eax中，作为当前函数的返回值。最后恢复寄存器内容并释放空间。    阅读完上述代码，我们发现func4是求斐波那契数列第x项的函数，参数代表递归层数。我们合理猜测程序需要我们输入一个数字，作为func4的参数，使得fucn4（斐波那契数列）的返回值等于0x37（换算成十进制为55）。  综上所述，我们发现斐波那契数列的第9项为55，故输入9即可过关。  启动GDB调试，查看0x401ec1处内容，发现确实要求输入一个整型数字。    第五关  阅读phase\_5代码。首先移动栈指针（开辟空间），将指针赋值给寄存器，将输入内容的地址存储到寄存器，将累加器清空。然后调用scanf函数，判断输入数据个数是否大于1，大于1则程序继续运行，否则触发炸弹。接下来将第一个数据赋值给寄存器%eax，%eax & 0xf的操作目的是将%eax限制在0～15的范围内（即第一个输入数据限制为0～15），再判断%eax是否为15，如果是则触发炸弹，否则程序继续运行。其次，将%ecx、%edx清零，准备开始循环，其中%edx为迭代计数器，要求迭代轮数为12轮；%ecx为累积器，要求每次跳转的地址处的值相加后等于第二个输入的数据。该循环是在一个表中不断跳转，初始位置即arr[%eax]。进入循环后，将M[%rax \* 4 + 0x401ba0]的数据搬移到%eax，然后将%eax的数据累加到%ecx中，再判断%eax和0xf是否相等，如果不相等则继续循环，否则结束循环。循环结束后，判断两个条件：迭代次数（%edx）是否等于12和累加数值（%ecx）是否等于第二个输入的数据，如果其中一个不满足则触发炸弹，否则过关。最后释放空间，第五关结束。    进入GDB调试查看0x401ba0处～0x401ba+15处的内容，得到下面的表格。   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 指针 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | | 数值 | 10 | 2 | 14 | 7 | 8 | 12 | 15 | 11 | 0 | 4 | 1 | 13 | 3 | 9 | 6 | 5 | | 来源 | 8 | 10 | 2 | 12 | 9 | 15 | 14 | 3 | 4 | 13 | 0 | 7 | 5 | 11 | 2 | 6 |   程序需要最后一步跳到指针6处，一共需要跳12步，因此起始点应该为7。将经过的点的数值累加起来得到93。举例，当前指针位6，来源为14，说明上一次指针为13，因为指针为13处的数值为6。  进入GDB调试，查看0x401ebe处的内容，发现确实是需要我们输入两个数字。    综上所述，我们只需要输入7和93即可通过关卡。  第六关  阅读phase\_6代码。程序首先申请空间，对寄存器赋初始值。然后将输入的值转化为8字节长整型存放在内存中，返回值是一个地址，通过不断地调用之后得到对应的%rax的值，如果最终的值与输入的%ebx的值不同则会触发炸弹。所以需要判断M[rax]是多少。    因为函数func中的代码逻辑过于晦涩难懂，所以我们使用GDB调试的得到我们需要的答案。首先在0x40110e处设置一个断点，然后输入r运行程序，随意输入一个数字。之后我们查看寄存器%rax内存的指针指向的地址的内容，发现此处的值为600。      综上所述，输入600即可过关。 |
| **五、实验结论：**  通过解析汇编代码的作用以及不断使用GDB对程序进行调试之后，最终通过了所有关卡（见下图）。 |

|  |
| --- |
| **六、心得体会：**   1. 通过本次试验，我对汇编代码有了更深刻的了解，知道如何使用汇编语言实现if语句，while语句，switch语句以及递归函数。 2. 通过本次试验，我了解了GDB一些常见的调试技巧该如何使用，包括p可以用于打印对应地址的信息，使用@+数字可以打印数组等。 |
| 指导教师批阅意见：  成绩评定：  指导教师签字：冯禹洪  2024年 月 日 |
| 备注： |