**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： 计算机与软件学院所有专业**

**指导教师： 刘 刚**

**报告人： 吴嘉楷 学号： 2022150168 班级： 国际班**

**实验时间： 2024年4月31日 ～ 2024年5月17日**

**实验报告提交时间： 2024年5月12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**  **反汇编二进制文件**  使用objdump反汇编工具对bomb\_64文件进行反汇编，并将结果输出到1.txt：    图1 反汇编指令  输入指令“gedit 1.txt”，以图形化方式打开1.txt文件，阅读汇编代码：    图2 1.txt文件概览  **第一关**  1.分析汇编代码内容  程序首先开辟了1个字节的栈空间，然后把地址0x401af8中的值作为第二个参数传递给strings\_not\_equal函数，如果函数的返回值是0，则会跳到地址400e87处执行指令，否则，就会调用explode\_bomb函数，使炸弹引爆。于是，我们需要探究如何使得strings\_not\_equal函数的返回值为0：  strings\_not\_equal函数的作用是比较两个参数中的字符串是否相等，若相等则返回0，不等则返回1，而参数1的字符串是我们输入的字符串，参数2的字符串是地址0x401af8中的值，因此，我们只需要输入地址$0x401af8中的字符串即可通过此关卡！  2.关键点：探究地址0x401af8中的字符串值  这里我们需要借助GDB调试工具。  首先，输入“gdb bomb\_64”开启gdb调试；  其次，输入“p (char \*) 0x401af8”查看地址0x401af8处的字符串值：    图3 借助gdb查看地址0x401af8处的字符串  3.破关  输入我们使用gdb查找到的地址0x401af8处的字符串“Science isn't about why, it's about why not?”：    图4 通过第一关  **第二关**  1.分析汇编代码内容  程序首先保存寄存器的值到栈上，然后将栈指针%rsp下移0x48个字节以开辟一段栈空间，作为数组的首地址，供后续读入六个数字使用，并将栈指针的值作为第二个参数传递给read\_six\_numbers函数，该函数会读入六个数字并存储在第二个参数指向的空间上方，若读取到的整数不大于5个，就会引爆炸弹，若**达到六个**则会恢复栈指针并返回。  然后程序更新基址指针寄存器的值为栈指针的值，即数组首地址，再把数组第三个元素的地址赋值给% r13寄存器，用于设置循环结束哨兵，把0赋值给% r12寄存器的低32位，用于累加元素的值。  其次，比较%rbp指向的地址中的数字与%rbp + 12指向的地址中的数字，若不相等则引发爆炸，否则将%rbp指向的地址中的数字累加到寄存器%r12中，再让%rbp + 4，检查%rbp指向的地址是否已经到了数组第四个元素的地址，若没到则继续循环比较和累加的操作，若到达则检查%r12中的值是否为0，为0则爆炸，否则恢复寄存器的值并返回。  2.简化内容  由于int类型所占的空间为4个字节，所以+12可以看作数组指针后移3位。  1. While（rbp != r13）:  2. If \*rbp != \*(rbp + 3)，then <explode\_bomb>！  3. \*(r12) += \*rbp  4. rbp += 1  5. If \*(r12) == 0 , then <explode\_bomb>！  因此，我们提取出此问题的三个要求：   1. 输入至少6个整数 2. arr [0] = arr [3] , arr[1] = arr[4] , arr[2] = arr[5] 3. arr[0] + arr[1] + arr[2] != 0   那么，该关卡将有无数种答案：  1 2 3 1 2 3 ， 0 1 0 0 1 0 ...  3.检验结果    图5.1 输入第一种测试答案    图5.2 输入第二种测试答案  甚至，输入7个数字也是可以的：    图5.3 输入7个数字的答案  **第三关**  1.分析汇编代码内容  程序首先分配了24个字节的空间用于存储从标准输入流读入的数据，然后分别将第三个参数值设为%rsp+12，将第四个参数值设为%rsp+8，把地址0x401ebe作为第二个参数传递给接下来调用的\_\_isoc99\_sscanf@plt输入函数。  我们猜测0x401ebe可能是一个字符串的首地址，于是可以通过gdb调试的方式查看字符串的值：    图6 查看0x401ebe处字符串的值  由得到的字符串“%d %d”，我们可推测：\_\_isoc99\_sscanf@plt函数应该是输入两个整数，把第一个整数的值存到第三个参数%rdx指向的地址（即%rsp+12）中，把第二个整数的值存到第四个参数%rcx指向的地址（即%rsp+8）中，然后返回读入整数的个数。  如果读入的整数个数不大于1，就会引发爆炸。因此，**条件1：输入两个整数。**  如果%rsp+12地址处的值大于7，就会引发爆炸。因此，**条件2：输入的第一个整数值小于大于7。**  然后，把%rsp+12处的值，即第一个输入的整数存到寄存器%eax中，然后无条件跳转到这个值\*8再**＋ 0x401b60**处存放的地址值指向的位置。于是，我们需要查看这些位置存放的地址值是什么。  由于我们输入的第一个整数范围为0 ~ 7，于是，我们需要查看0x401b60、0x401b68、...、0x401b98 8个位置存放的地址值：    图7 查看存放的地址值  如果第一个整数输入0，则会跳转到0x00400f32处，将0x217即535存到寄存器%eax中，然后将寄存器的值与我们输入的第二个整数值进行比较，若不相等，则会引发爆炸。  依照上述思路，我们可以分析出输入的第一个整数与第二个整数的对应关系。  因此，我们分析得到**条件3：（条件3已经满足了条件1、2，即为答案）**  **若第一个整数为0，则第二个整数应该为535；**  **若第一个整数为1，则第二个整数应该为926；**  **若第一个整数为2，则第二个整数应该为214；**  **若第一个整数为3，则第二个整数应该为339；**  **若第一个整数为4，则第二个整数应该为119；**  **若第一个整数为5，则第二个整数应该为352；**  **若第一个整数为6，则第二个整数应该为919；**  **若第一个整数为7，则第二个整数应该为412；**  2.检验结果    图8 输入“7 412”可以通过第三关  **第四关**  1.分析汇编代码内容  程序首先在栈上分配0x18即24个字节的空间，并把寄存器%rsp+12的地址值存储在寄存器%rdx中作为第三个参数传递，把地址0x401ec1加载到%esi中作为第一个参数传递给\_\_isoc99\_sscanf@plt函数，我们猜测地址0x401ec1可能是格式字符串的首地址，并使用了GDB调试工具查看：    图9 地址0x401ec1的字符串值  由查看结果可见，该字符串是“%d”，所以\_\_isoc99\_sscanf@plt函数应该是需要读入1个整数，并将读入的整数值存放在%edx指向的地址，即%rsp+12处，然后返回读入的整数个数。  若读入的整数个数不等于1，就会引发爆炸！因此，**条件1：输入1个整数。**  接着，程序比较%rsp+12处的值，即输入的整数的值与0的关系，若输入的整数小于等于0，就会引发爆炸，因此，**条件2：输入的整数大于0。**  然后，将该整数作为第一个参数传递给函数func4，并比较func4的返回值是否等于0x37，即55，若不等于，则会引发爆炸！  于是，我们需要探究如何使func4的返回值等于55：  函数首先将1赋值给寄存器%eax，确保返回值至少为1，然后比较%edi的值是否小于等于1，若是，则恢复寄存器的值并返回；否则，将第一个参数的值减1并进行递归，将递归结果存储到%ebp中，再将第一个参数的值减2并进行递归，得到的结果加上%ebp中的结果作为函数的返回值返回。这不得不让我们想起斐波那契数列的递归公式：f（n）=f（n-1）+f（n-2）。  2.简化func4函数内容  func4（n）：  If（n<=1) return 1;  return func4(n-1) + func4(n-2);  因此，我们可以得到输入的数字n为斐波那契数列的（项数-1）。因为斐波那契数列前几项为：1 1 2 3 5 8 13 21 34 55 89，从中我们发现55是数列的第10项，于是我们应该输入10-1 = 9，即可使func4返回55！  因此，**条件3：输入整数9。（条件3已经包含了前两个条件，故可作为最终条件）**  3.检验结果    图10 输入9可通过第四关  **第五关**  1.分析汇编代码内容  程序首先分配了24个字节的空间用于存储从标准输入流读入的数据，然后分别将第三个参数值设为%rsp+12，将第四个参数值设为%rsp+8，把地址0x401ebe作为第二个参数传递给接下来调用的\_\_isoc99\_sscanf@plt输入函数，同时把寄存器%eax初始化为0。  我们猜测0x401ebe可能是一个字符串的首地址，于是可以通过gdb调试的方式查看字符串的值：    图11 查看0x401ebe处字符串的值  由得到的字符串“%d %d”，我们可推测：\_\_isoc99\_sscanf@plt函数应该是输入两个整数，把第一个整数的值存到第三个参数%rdx指向的地址（即%rsp+12）中，把第二个整数的值存到第四个参数%rcx指向的地址（即%rsp+8）中，然后返回读入整数的个数。  如果读入的整数个数不大于1，就会引发爆炸。因此，**条件1：输入两个整数。**  然后，程序把%rsp+12处的值，即我们输入的第一个整数存储在寄存器%eax中，然后对这个整数的二进制值取低4位，即对该整数取模16，得到一个0 ~ 15的值，并更新地址中的值，若结果等于15，则会引爆炸弹！因此，**条件2：第一个整数%16 ！= 15。**  其次，程序初始化寄存器%ecx和%edx的值为0，让%edx的值+1（充当计数器作用），然后把%eax的值更新为0x401ba0+4\*（%eax）地址处的值，然后将结果累加到寄存器%ecx中，再比较%eax的值是否等于15，如果不等于，就循环让%edx的值+1，再更新%eax，再比较它与15的关系，直到%eax等于15为止。  于是，我们需要知道0x401ba0+4\*（%eax）地址处的值可以是什么，我们猜测0x401ba0可能是某个数组的首地址，数组中存在一个元素15作为循环的退出条件。为了验证我们的猜想，我们通过GDB调试来查询0x401ba0地址之后的16个元素（由%16的操作我们推测%eax的值不大于16）：    图12 查询0x401ba0地址处的数组  由查询结果可见，数组中确实存在一个元素值为15。数组中的元素值在循环过程中是充当一个偏移量的，基址是数组的首地址0x401ba0。且偏移量不会超出该数组的范围。  接着，程序将走出循环的%eax即15的值存储到%rsp+12的地址处，再比较循环计数器%edx的值与12的关系，若%edx的值不等于12，则会引发爆炸！因此，**条件3：循环的次数必须为12。**  最后，程序还会比较%rsp+8地址中的值与%ecx的值，即比较我们输入的第二个整数与元素累加得到的值，若二者不相等，就会引发爆炸。因此，**条件4：第二个整数 = 循环12次的元素累加值。**  2.明确要求  输入的第一个整数会使得循环执行12次并在第12次指向的元素值为15，范围为0~14。  输入的第二个整数等于循环12次的元素累加值。  于是，关键点在于输入什么数字能使循环恰好执行12次，而循环执行的次数与0x401ba0处的数组息息相关，因此，我们将数组的信息提取成一个表格，方便我们进行探究：    图13 数组信息（上：下标，下：数值）  接着，我们尝试输入不同的数字，使循环次数刚好为12次：  （1）如果输入0，那么循环过程为：0 -> 10 -> 1 -> 2 -> 14 -> 6 -> 15 ，循环次数共6次，小于12次。  并且循环过程中出现的数字1，2，6，10，14，15也将被排除，因为输入这些数字的循环次数必然会小于6。  （2）如果输入3，那么循环过程为：3 -> 7 -> 11 -> 13 -> 9 -> 4 -> 8 -> 0 -> 10 -> 1 -> 2 -> 14 -> 6 -> 15 , 循环次数共13次，大于12。  我们只需要让循环次数减1就可以达到要求了，最直接的方式就是从本次循环的第二个点开始就可以了。分析可知，循环的第二个点为7，因此，我们**输入第一个整数值为7**就可以满足要求。  再者，我们计算循环12次的元素的累加和：  11 + 13 + 9 + 4 + 8 + 0 + 10 + 1 + 2 + 14 + 6 + 15 = **93**  因此，我们得出答案：**7 93**  3.检测结果    图14 输入“7 93”可以通过第五关  **第六关**  1.分析汇编代码内容  程序首先分配一段8个字节的空间，然后将10赋值给%edx作为接下来要调用的fun6的第三个参数，将0赋值给%esi作为第二个参数，然后调用strtol@plt函数，读入我们输入的字符串并转换为long类型（占8个字节），并将返回值（转化后的整数）存储在0x602780地址处，又将该地址加载到寄存器%edi作为第一个参数传入fun6，接着调用了fun6函数。  然后，将fun6函数返回值（应该是一个指针）+8地址处的值加载到%rax中，重复执行此操作3次。由于long类型（或指针）占8个字节，加8的偏移量应该是偏移越过一个long类型的数据（或一个指针），然后不断从偏移结果里面取数据，作为下一次跳转的基址，这不得不令我们联想到了链表：    图15 链表节点的跳转  于是，我们猜测fun6函数可能会对0x602780为头节点的链表执行某些操作，然后返回链表某个节点的地址指针。  接下来，我们需要探究fun6函数的执行过程：fun6（0，0x602780，10）  首先fun6函数将%rdi+8即0x602788处的值保存在%r8中，并清空0x602788处的值，然后保存%rdi寄存器的值至%rax和%rcx，检验%r8即0x602788处的值是否为0，为0则返回%rdi的值。后续的代码涉及多次跳转，实在难以理解，无法直接分析出fun6函数的作用。于是，我打算使用GDB调试工具在phase\_6函数执行fun6前后打断点，查看fun6带来的影响。  首先，在fun6执行前，即0x4010f7处打断点，在fun6执行后，即 0x4010fc处打第二个断点：    图16 设置断点  然后，我们运行程序，在fun6执行前输入参数0，查看链表各节点（最后一节点指针域为空）的数据域和指针域：    图17 fun6执行前的链表数据  接着，我们输入“c”指令继续程序，来到第二个断点处，查看链表：    图18 fun6执行后的链表数据  由链表中的数据变化可知，链表各节点的连接顺序发生了改变，使得新顺序下数据域的数据变得有序了，因此，我们可以推断出：fun6函数的作用就是根据节点数据域的值对链表进行降序排序，然后返回最大值所在节点（头节点）的地址。  拿到该地址后，程序将往后寻址三次，即找到链表的第四个节点（第四大），然后比较该节点数据域的值是否与我们输入的数字相等，若不相等，则会引发爆炸！  2.明确要求  输入的数字 = 排序后的第四大数值  由排序后序列可见，当我们输入0时，第四大的数为600，因此我们可以输入600。  此外，因为我们输入的数字也会加入排序序列，因此，我们可以输入一个排序后会成为第四大的数字，即可满足要求，因为第三大数为673，第四大数为600，在此情形下，我们可以输入【600，673】。  3.检验结果    图19.1 输入600可以通过第六关    图19.2 输入673也可以通过第六关  **隐藏关卡**  1.观察主函数，发现过关检测是由phase\_defused函数检验的，于是，phase\_defused函数隐含了到达隐藏关卡的“密钥”。    图20 phase\_defused函数的关键信息  由上图可见，函数会调用一个string\_not\_equal函数检验输入的字符串是否等于地址0x401eca处的字符串，如果等于，string\_not\_equal函数会返回0，使得跳转无法执行，如果不等于，则会返回非0，跳过secret\_phase隐藏关卡的启动。  那么，这个字符串应该在什么时机输入呢？  2.分析phase\_defused函数，开始与6进行了比较，推测过了第六关才能开启隐藏关卡：    图20.1 隐藏关卡开启时机推测  然后，我们定位到函数中的一些关键参数：    图20.2 关键参数  从cmp指令我们大致推断需要输入两个数据才能开启隐藏关卡，并使用了GDB调试工具查看了相关参数的信息：    图20.3 相关参数信息  由此，我们可以推断出，在关卡1~5中，我们需要输入**一个整数 ＋ “austinpowers”**作为答案，才可以打开隐藏关卡！  观察我们的通关答案，我们可以发现，**“austinpowers”**应该是一个隐藏关卡的开启密钥，需要在我们某次过关时添加上，而满足仅输入一个整数的只有 **关卡四** ！  因此，我们需要在关卡四过关时追加答案 **austinpowers 。**  3.检测是否可以开启隐藏关卡    图21 找到了隐藏关卡  4.分析隐藏关卡  程序首先要求我们输入一行数据，然后调用strtol@plt函数将字符串转化为long类型的整数，将数值-1与0x3e8即1000进行比较，若小于等于则继续执行，否则引发爆炸。因此，**条件1：输入 <= 1001**。  接着，将我们输入的整数赋值给%esi作为第二个参数，将地址0x6025a0赋值给%edi作为第一个参数，然后调用fun7函数，若fun7返回值不等于3，则会引爆炸弹！  于是，我们需要探究如何使fun7函数的返回值等于3：  经过探查，我们发现地址0x6025a0的值为36：    图22.1 gdb查看地址0x6025a0的值  fun7函数首先检测%rdi的值是否为0，如果是，则直接返回-1；  否则，将%edi处的值(一开始是36)加载到寄存器%edx中，比较%esi和%edx，如果%edx > %esi，就将%rdi的值+8，递归地调用自己，然后将递归结果\*2，返回最终结果；反之，若%esi = %edx，直接返回0，若%esi < %edx, 就将%rdi+16，然后将递归结果\*2 + 1，返回。  5.化简为伪代码  fun7( rdi , rsi):  If rdi == 0, return -1;  If rsi < \*(rdi) , return 2 \* fun7( \*(rdi + 8) , rsi )  Else  If rsi == \*(rdi) , return 0;  Else return 2 \* fun(\*( rdi + 16) , rsi ) + 1  由上述伪代码，我们容易联想到二叉搜索树，左子树数据小于36，根节点数据为36，右子树数据大于36，rsi为所要搜索的节点的数据域数值，rdi为节点的地址指针，返回值为所要搜索的二叉树节点的序号，（从0开始）。  要使返回值为3，即要搜索第4个节点，如下图所示：    图23 二叉搜索数简图  因此，我们需要知道该节点数据域的数值是多少，使用gdb查看：    图24 一直往右子节点搜索  由图可见，输入107即为答案！  6.检验结果    图25 输入107通过隐藏关卡 |
| **五、实验总结：**  1.初步掌握了objdump反汇编工具的使用方法：使用“objdump -d 源二进制文件 > 目标文件”指令可以把源二进制文件反汇编输出为目标文件。  2.更加熟悉了GDB调试工具的使用：  “gdb 目标文件”——启用gdb调试工具调试目标文件  “b \*0xfff”——在地址0xfff处打断点  “r”——开始执行  “p”——打印  “p （char \*） 0xff”——显示首地址为0xff的字符串的值  “x 0xff”——显示地址0xff处存放的值  “p \*0xff @n”——显示从地址0xff开始的n个位置存放的数值  “/x”——以十六进制格式  “p /x”——以十六进制形式打印  “p \*（int\*） 0xff”——打印处地址0xff处存放的整形值  “c”——继续往下执行到下一个断点处  3.对汇编指令的掌握更加深刻：  （1）jmp \*0x401b60(,%rax,8)与jmp 0x401b60(,%rax,8)的区别在于加\*可以取出计算结果中的间接地址，相当于一次解引用操作。  （2）cmp指令是第二个操作数比第一个操作数，如cmp %edi, %esi ，若%esi > %edi，则标志位会根据大于的结果置数。  （3）ja和jg都是大于的时候跳转，jb和jl都是小于的时候跳转。  （4）%rbx是%ebx的64位扩展。  （5）cltq指令会将eax寄存器中的32位有符号整数符号扩展到64位，并将这个扩展后的值存储在rax寄存器中，并且将扩展的值存到edx寄存器中。  （6）test指令可以用于判断操作数是否为0  4.答案汇总  **第一关**  Science isn't about why, it's about why not?  **第二关**  1 2 3 1 2 3  或  0 1 0 0 1 0  ...  **第三关**  0 535  1 926  2 214  3 339  4 119  5 352  6 919  7 412  **第四关**  9 or 9 austinpowers(可开启隐藏关卡)  **第五关**  7 93  **第六关**  600 ~ 673 都可以  **隐藏关卡**  107 |

|  |
| --- |
| **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： |