

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**专业班级： 校交1601**

**学 号： U201612696**

**姓 名： 陈淏睿**

**指导教师： 石宣化**

**报告日期： 2018年6月 20日**

**计算机科学与技术学院**

# 

# 实验2： 拆弹实验

**2.1 实验概述**

实验目的：增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。

实验目标：需要拆除尽可能多的炸弹。

实验要求：使用gdb调试器和objdump来反汇编炸弹的可执行文件，并单步跟踪调试每一阶段的机器代码，从中理解每一汇编语言代码的行为或作用，进而设法“推断”出拆除炸弹所需的目标字符串。

实验语言：c。

实验环境：linux

报告写作环境：windows

**2.2 实验内容**

一个“binary bombs”（二进制炸弹，下文将简称为炸弹）是一个Linux可执行C程序，包含了6个阶段（phase1~phase6）。炸弹运行的每个阶段要求你输入一个特定的字符串，若你的输入符合程序预期的输入，该阶段的炸弹就被“拆除”，否则炸弹“爆炸”并打印输出 "BOOM!!!"字样。实验的目标是拆除尽可能多的炸弹层次。

每个炸弹阶段考察了机器级语言程序的一个不同方面，难度逐级递增：

\* 阶段1：字符串比较

\* 阶段2：循环

\* 阶段3：条件/分支

\* 阶段4：递归调用和栈

\* 阶段5：指针

\* 阶段6：链表/指针/结构

另外还有一个隐藏阶段，但只有当你在第4阶段的解之后附加一特定字符串后才会出现。

为了完成二进制炸弹拆除任务，你需要使用gdb调试器和objdump来反汇编炸弹的可执行文件，并单步跟踪调试每一阶段的机器代码，从中理解每一汇编语言代码的行为或作用，进而设法“推断”出拆除炸弹所需的目标字符串。这可能需要你在每一阶段的开始代码前和引爆炸弹的函数前设置断点，以便于调试。

**2.2.1 阶段1 字符串比较**

1. 任务描述：通过phase\_1的反汇编代码找出要输入的字符串。
2. 实验设计：利用gdb结合断点来动态地分析。
3. 实验过程：

观察phase\_1的反汇编代码，如图2.1.1所示：

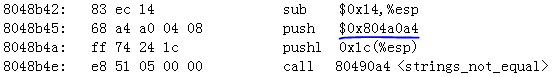


图2.1.1

发现在调用strings\_not\_equal对比字符串之前，有一个地址送入了堆栈，猜测该地址就是正确字符串的首址，于是直接在gdb中用p命令查看字符串。如图2.1.2所示：



图2.1.2

故猜测“He is evil and fits easily into most overhead storage bins”就是所需字符串。重新执行该程序，直接输入该字符串，发现拆弹成功。如图2.1.3所示：



图2.1.3

**2.2.2 阶段2 循环**

1. 任务描述：通过phase\_2的反汇编代码推断第二阶段要输入的数据
2. 实验设计：利用gdb结合断点分析
3. 实验过程：

观察phase\_1的前一部分反汇编代码，如图2.2.1所示：

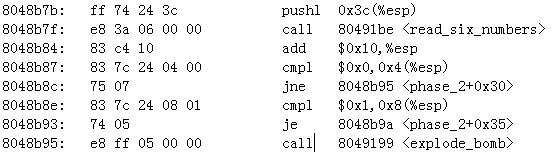


图2.2.1

发现输入的应该是六个数字，而且要求第一个数字为1，第二个数字为2.继续向后观察代码，如图2.2.2所示，发现这是一个循环，ebx作为指向长为6的数组的指针，首先指向第一个元素0，之后以后一元素与前两个元素之和是否相等作为判断条件，即6个数字构成Fibonacci数列：0,1,1,2,3,5.

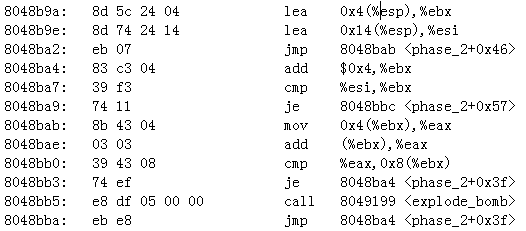


图2.2.2

重新执行该程序，输入“0 1 1 2 3 5”，发现拆弹成功。如图2.2.3所示：



图2.2.3

**2.2.3 阶段3 条件/分支**

1. 任务描述：通过phase\_3的反汇编代码推断第三阶段要输入的数据
2. 实验设计：利用gdb结合断点来动态地分析
3. 实验过程：

根据阶段1的经验调用sscanf函数之前压栈的地址0x804a106将作为函数的参数，如图2.3.1所示：

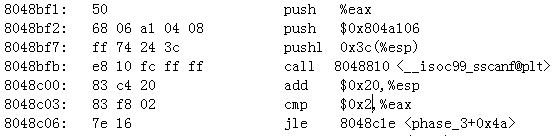


图2.3.1

因此将该地址放进gdb中查看内容，结果如图2.3.2所示，该函数要求输入1个int,1个char和1个int，图2.3.1中说明参数≤2时会跳转至explode\_bomb引爆炸弹，侧面证实参数应为3个：



图2.3.2

继续观察phase\_3的后一部分反汇编代码，如图2.3.3所示：

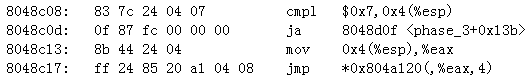


图2.3.3

显然，一个数字要求不能比0x7大，同时出现了\*0x804a120(,%eax,4)跳转表标志性语句，证明之后是一个switch结构。继续向下观察代码，之后的代码分别对应第一个输入为0~7时第二，第三个输入的情况，其中第二个数看做ASCII码，第三个数应转换为十进制。如图2.3.4-2.3.11所示。



图2.3.4



图2.3.5



图2.3.6



图2.3.7



图2.3.8



图2.3.9



图2.3.10



图2.3.11

只选第一个输入为0和1时进行验证，如图2.3.12和2.3.13所示，证明第三阶段拆弹成功：



图2.3.12



图2.3.13

**2.2.4 阶段4 递归调用和栈**

1. 任务描述：通过phase\_4以及func4的反汇编代码推断第四阶段要输入的数据
2. 实验设计：利用gdb结合断点来动态地分析
3. 实验过程：

观察phase\_4的反汇编代码，如图2.4.1所示，发现再次出现sscanf前的地址压栈，根据阶段3的经验判断同时之后的代码说明给sscanf的参数为2.：

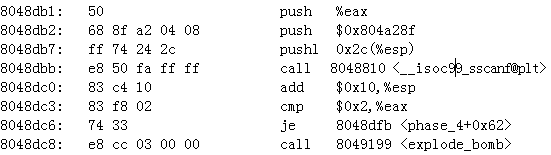


图2.4.1

在gdb中检查上述地址的内容，结果如图2.4.2所示，说明是两个int：



图2.4.2

按着程序的跳转，跳过fnc4出现的位置向后看发现要求输入的第一个int≤14，第二个int确定为4，如图2.4.3所示：

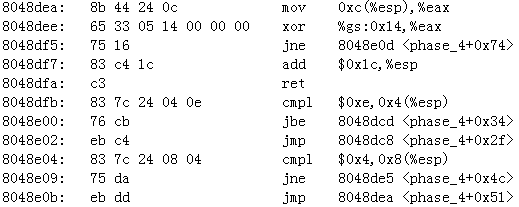


图2.4.3

由于第一个输入的范围以确定，因此在此处不妨直接用枚举试探bomb，试探到输入为2 4时输出正确结果，如图2.4.4所示：



图2.4.4

**2.2.5 阶段5 指针**

1. 任务描述：通过phase\_5的反汇编代码推断第五阶段要输入的数据
2. 实验设计：利用gdb结合断点来动态地分析
3. 实验过程：

观察phase\_5的前一部分反汇编代码，如图2.5.1所示，发现string\_length函数的输出应为6，即此阶段应有6个输入：

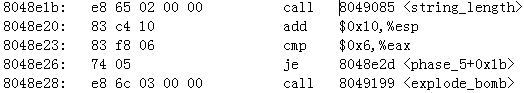


图2.5.1

观察下一部分代码，如图2.5.2所示，发现这一部分是一次数为6的循环，将6个输入依次与0xf与后的值作为偏移地址在0x804a140为首址的表中查找元素并求和，其和为0x18：

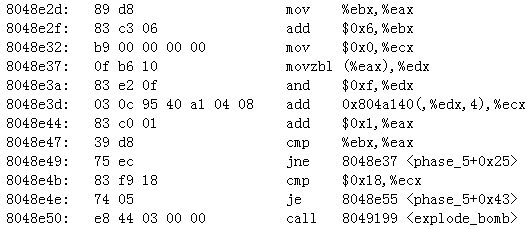


图2.5.2

在gdb中观察上述地址，如图2.5.3所示。由于只需6个数之和为24而对每个数的具体值无要求，因此索性令6个数均为4，其相对0x804a140偏移地址为32，比例因子为4，故偏移地址为8：

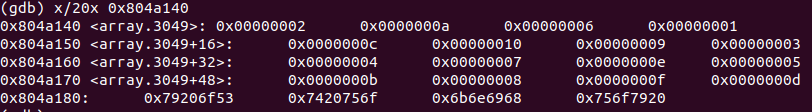


图2.5.3

查ASCII码表有h的ASCII码为68，输入6个h，发现拆弹成功，如图2.5.4所示：



图2.5.4

**2.2.6 阶段6 链表/指针/结构**

1. 任务描述：通过phase\_6的反汇编代码推断第五阶段要输入的数据
2. 实验设计：利用gdb结合断点来动态地分析
3. 实验过程：

观察phase\_6的前一部分反汇编代码，如图2.6.1所示，发现本阶段需要输入6个数，同时要求减去以后必须不能大于5，即要在1、2、3、4、5、6中的数字才可以：

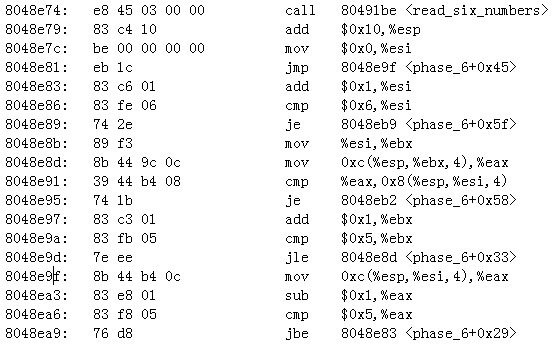


图2.6.1

继续看phase\_6的下一段反汇编代码，不难发现这是通过esi与ebx来实现的双重循环，一一比较输入的值是否相同，只要有一对相同该炸弹就会爆炸，所以输入的数据应该是1到6的一种排列才可以。如图2.6.2所示：

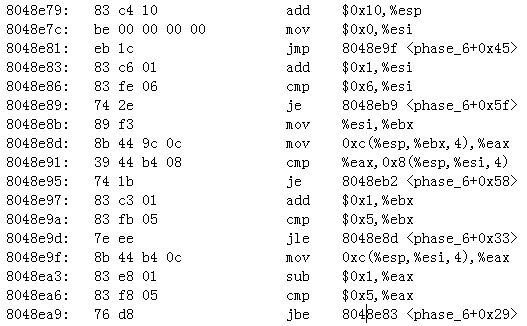


图2.6.2

继续看下一段代码。这一段又是一个循环，将输入的x变为7-x,如图2.6.3所示。

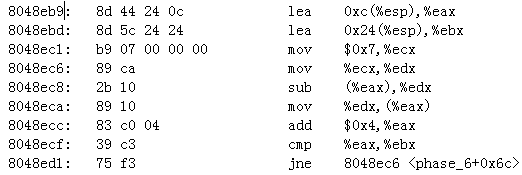


图2.6.3

继续看下一段反汇编代码，其中又出现了地址。如图2.6.4所示。

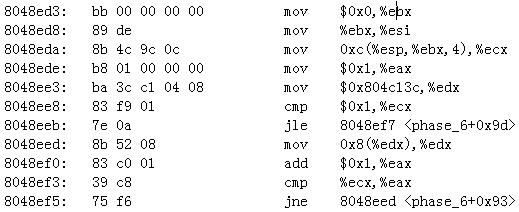


图2.6.4

在gdb中观察该地址，发现其之后的一段空间为链表形式，符合该阶段提示。链表中每个结点有三个元素，分别为数值，序号和指向下一结点的指针，如图2.6.5所示：

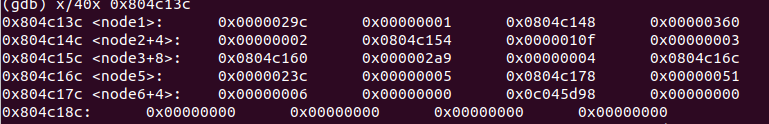


图2.6.5

观察该阶段最后一段代码，如图2.6.6所示，发现其操作为检查链表中元素是否降序排序，故我们输入的6个小于7的数应在被7减之后组成新的序列，与该序列符合的结点数值顺序为降序。降序排序为2(0x360),4(0x2a9),1(0x29c),5(0x23c),3(0x10f),6(0x51),故输入应为5 3 6 2 4 1.

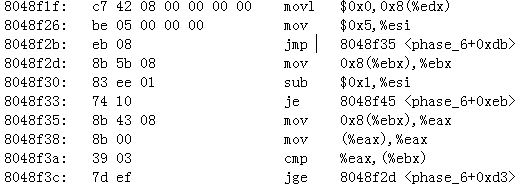


图2.6.6

重新运行程序，输入该字符串进行测试，如图2.6.7所示：



图2.6.7

**2.2.7 阶段7 隐藏阶段**

1. 任务描述：找出隐藏阶段开启方式并且拆除隐藏阶段的炸弹。
2. 实验设计：利用gdb结合断点来动态地分析
3. 实验过程：

直接在反汇编代码中观察发现，只有phase\_defused函数中有对secret\_phase函数的调用，于是观察phase\_defused函数的反汇编代码，如图2.7.1所示，发现此处出现一地址，在gdb中查看可知意义为输入六个字符串，即完成6个phase的拆弹后满足此处条件，如图2.7.2所示：

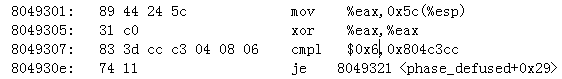
 图2.7.1



图2.7.2

按跳转向下观察，发现sscanf需要有三个参数才能跳转，且结合提示得知其具体形式为phase4的解答后跟一字符串，如图2.7.3所示：

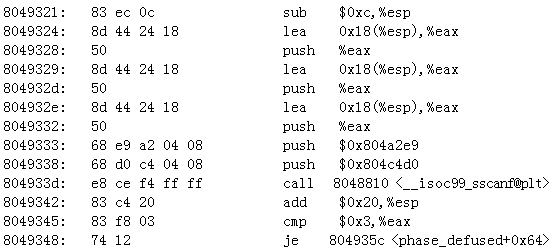


图2.7.3

继续观察，发现压栈一个地址后调用了strings\_not\_euqal进行字符串比较，如图2.7.4所示，猜想地址中存放了目标字符串。

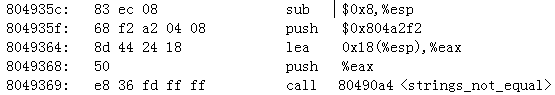


图2.7.4

将该地址放入gdb观察，得到字符串DrEvil,如图2.7.5所示：



图2.7.5

在解密phase4时加入该字符串，完成phase6后发现弹出提示：找到secret\_phase,如图2.7.6所示：

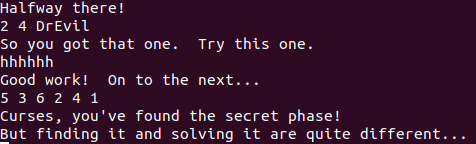


图2.7.6

观察secret phase的代码，发现phase的输入为一数值，且能不大于0x3e8+1=1001。Phase中调用了fun7,两参数中一个是输入值，另一个为地址；返回值必须为2，如图2.7.7所示：

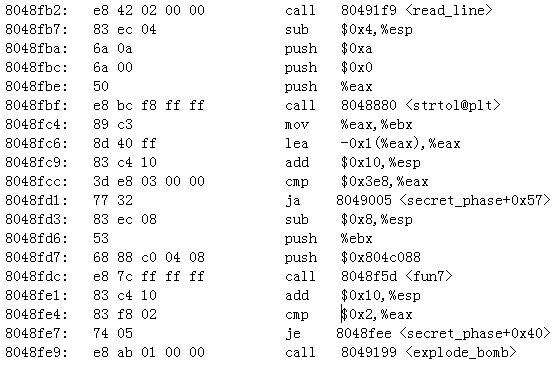


图2.7.7

fun7的跳转有些乱，但分析后可发现其中有递归调用，第一次调用时其中edx，ecx依次为0x804c088，x（为输入的数）。然后比较edx与0，如果相等就返回-1，否则ebx=[edx],然后比较ebx与ecx，也就是根据传入fun7的第一个参数处的数值与第二个参数比较结果进行相应的处理。

当第一个参数处的数值小于第二个参数时，就依次将原来的第一次参数加上0x8处的值、原来的第二个参数作为参数调用fun7，然后将返回值乘2加1作为返回值输出。即原来参数依次为a、b，传入参数依次为[a+0x8]、b；

当第一个参数处的数值大于第二个参数时，就依次将原来的第一次参数加上0x4处的值、原来的第二个参数作为参数调用fun7，然后将返回值乘以2作为返回值输出。即原来参数依次为a、b，传入参数依次为[a+0x4]、b；

当第一个参数等于两个参数的平均值时，就直接将平均值置为0输出。如图2.7.8所示：

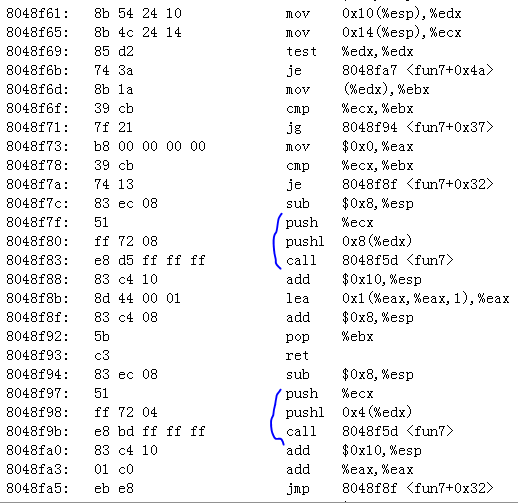


图2.7.8

由于返回值为2，因此采用的是乘2返回的递归方式，应到0x804c088+0x4=0x804c08c中查看。在gdb中观察0x804c088开始的内容，如图2.7.9所示，发现0x804c08c中仍为一地址：

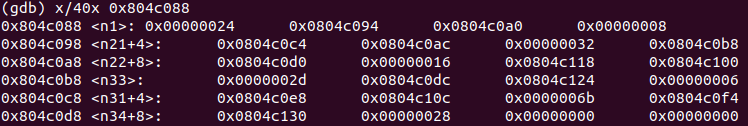


图2.7.9

将该地址放入gdb观察，发现其值为八进制\026,如图2.7.10所示。转换为十进制为22.



图2.7.10

所以输入的值应该为22，故进入secret phase后输入22，破解成功，如图2.7.11所示：

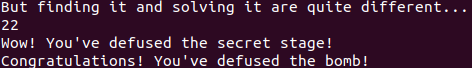


图2.7.11

**1.3 实验小结**

这一次的二进制炸弹实验相较于以前的各种实验来说更加有趣和新颖，当然难度也显著提高。这次实验主要运用的就是逆向方面的技术，汇编语言中一些细节的遗忘和实验环境下汇编语言的形式与上课有所不同带来的陌生感为我的拆弹带来了一定的阻碍，堆栈传递参数的密集出现对我而言也是一个不小的挑战。

我认为这次实验的关键之处在于找到输入数据的流向，输入后存在哪里，之后进行了哪些操作，这都是解决拆弹的核心之处。

通过这次实验，我对递归调用的手段了解的更加了深入，虽然递归的函数要求必须基本看懂递归的功能，花费了大量的时间，但是这也是对我耐心的一种培养，逆向工程没有耐心是永远做不好的。

总之这次实验的过程整体还是很愉快的，虽然耗时比较长但是收获也很大！