

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**专业班级： 计卓1601**

**学 号： U201610282**

**姓 名： 幸儒林**

**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**

目录

[实验2： Binary Bomb 1](#_Toc517983087)

[2.1 实验概述 1](#_Toc517983090)

[2.1.1 实验目的意义 1](#_Toc517983091)

[2.1.2 实验目标 1](#_Toc517983092)

[2.1.3 实验要求 1](#_Toc517983093)

[2.1.4 实验安排 1](#_Toc517983094)

[2.2 实验内容 2](#_Toc517983095)

[2.2.1 阶段1: 字符串比较 4](#_Toc517983096)

[2.2.2 阶段2: 循环 6](#_Toc517983097)

[2.2.3 阶段3: 条件/分支 9](#_Toc517983098)

[2.2.4 阶段4: 递归调用和栈 13](#_Toc517983099)

[2.2.5 阶段5: 指针 18](#_Toc517983100)

[2.2.6 阶段6: 链表/指针/结构 23](#_Toc517983101)

[2.2.7 阶段7: 隐藏阶段 29](#_Toc517983102)

[2.3 实验小结 38](#_Toc517983103)

# 实验2： Binary Bomb

2. 1. 实验概述
      1. 实验目的意义

本次实验中，需要使用课程所学知识拆除一个“**binary bombs**”来增强对程序机器级表示、汇编语言、调试器、逆向工程等方面原理和技能的掌握。

* + 1. 实验目标

完成6个阶段的炸弹拆除。并发现隐藏阶段完成对于隐藏阶段的拆除。

* + 1. 实验要求

在Linux系统环境下使用objdump反汇编工具实现对于目标程序的反汇编，通过得到的反汇编代码获知程序运行结构以及一些其他有用信息，通过反汇编工具查看程序内部变量等数据或通过gdb改变程序执行环境，通过以上两个工具完成对于炸弹的拆除。

* + 1. 实验安排

本次实验中，会得到一个不同的 binary bomb 二进制可执行程序及其相关文件，其中包含如下文件：

* bomb：bomb 的可执行程序。
* bomb.c：bomb 程序的 main 函数。

运行./bomb：./bomb 是一个可执行程序，需要 0 或 1 个命令行参数（详见 bomb.c 源文件中的 main()函数）。如果运行时不指定参数，则该程序打印出欢迎信息后，期待按行输入每一阶段用来拆除炸弹的字符串，并根据当前输入的字符串决定是通过相应阶段还是炸弹爆炸导致任务失败。

也可将拆除每一阶段炸弹的字符串按行组织在一个文本文件中（比如： ans.txt），然后作为运行程序时的唯一一个命令行参数传给程序（./bomb ans.txt），程序会自动读取文本文件中的字符串，并依次检查对应每一阶段的字符串来决定炸弹拆除成败。

* 1. 实验内容

通过所学知识完成拆除“binary bombs”。

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。

实验语言：C 语言

实验环境：linux

在正式开始实验前需要进行一些准备工作:

1. 完成对于objdump，gdb的安装。

在Ubuntu终端下输入以下安装指令完成拆弹工具objdump，gdb安装

sudo apt-get install objdump

sudo apt-get install gdb

1. 完成对于炸弹程序的反汇编

利用反汇编工具实现对于炸弹程序的反汇编



图2-2-0-1 objdump反汇编

这样我们就可以来查看反汇编代码了，用Sublime Text 3打开如下：

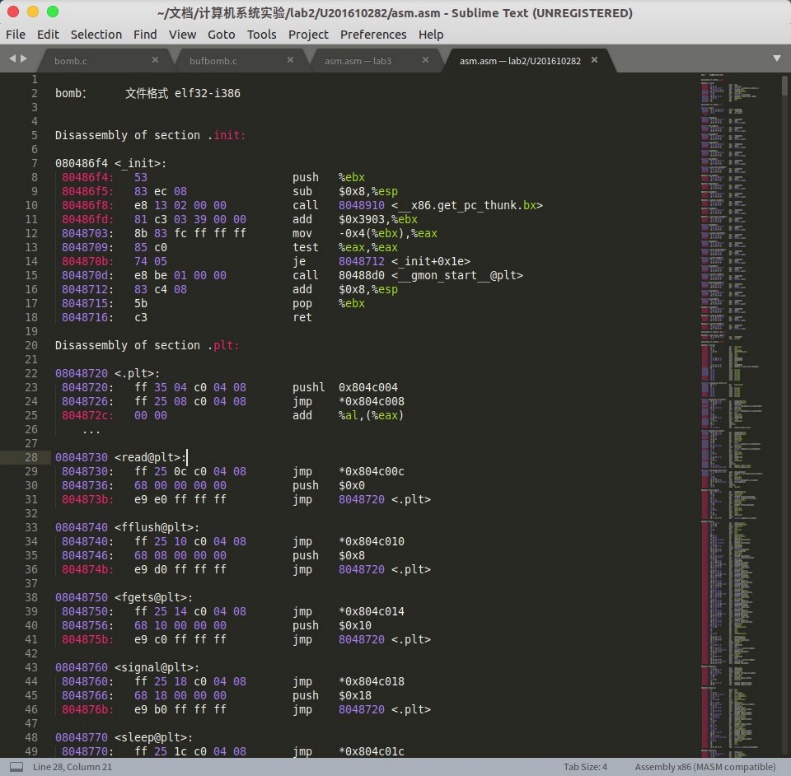


图2-2-0-1 sublime text 3打开反汇编状态图

下面将分7个阶段详细介绍对于炸弹的拆除过程。

* + 1. 阶段1: 字符串比较

1. 任务描述

本阶段主要考察的是字符串比较，需要通过反汇编工具objdump完成对于程序的反汇编并根据反汇编代码来完成，这一步在之前的准备工作中已经做了，下面的任务就是分析反汇编代码，利用gdb工具获取相应信息，将正确的输入字符串找出来。

1. 实验设计

本阶段实验需要找到正确的字符串并将之输入，所以我们需要在反汇编代码中找到对应的字符串比较代码，然后分析其是和哪一个字符串比较的(字符串地址)，然后用gdb运行该炸弹程序并查询对应地址下的字符串，这样就可以得到正确的字符串了。

1. 实验过程

实验过程分为三步：

1. 查看并分析反汇编代码，找到正确字符串首地址

首先找到phase\_1反汇编代码

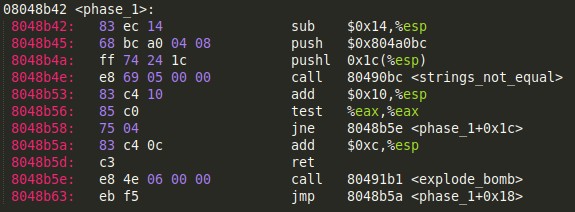


图2.2.1-1 phase\_1反汇编代码

我们从中可以得到phase\_1的操作流程，

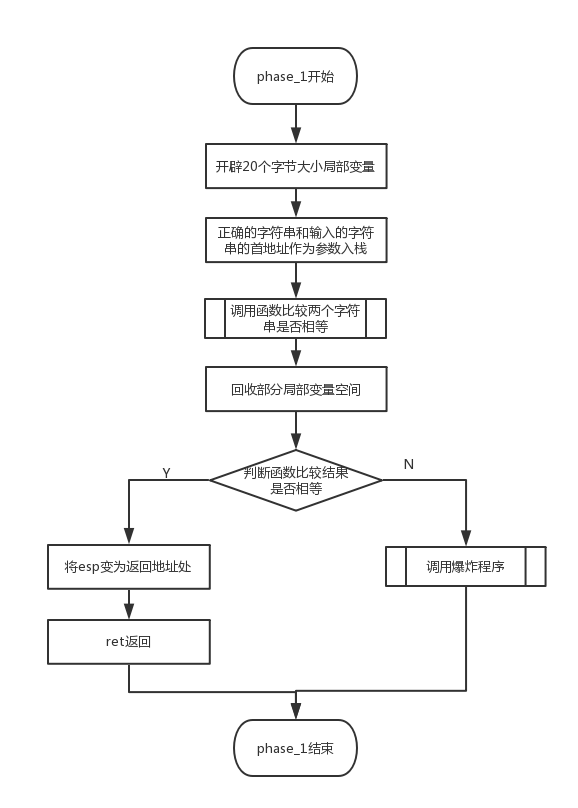


图2.2.1-2 phase\_1流程图

从中我们可以得知正确字符串地址为0x804a0bc

1. 运用gdb调试炸弹程序，并查看该字符串内容



图2.2.1-3 phase\_1 gdb查看正确字符串图

可以得到正确字符串为:

When I get angry, Mr. Bigglesworth gets upset.

1. 在破解文件中输入正确字符串实现对于阶段1炸弹的破除

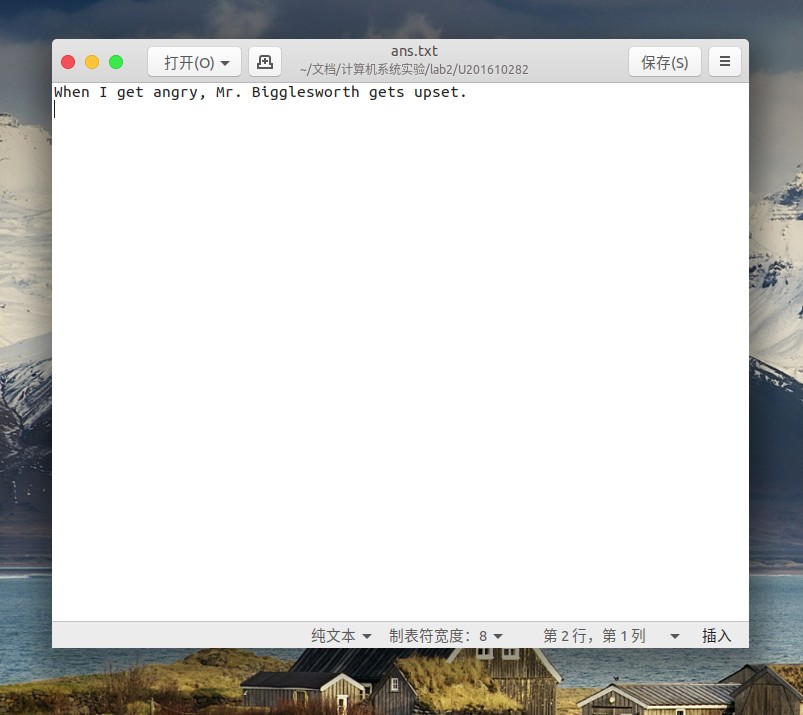


图2.2.1-4 phase\_1 完成后结果文件图

1. 实验结果

在终端中将结果文件作为输入运行炸弹程序，成功破解！

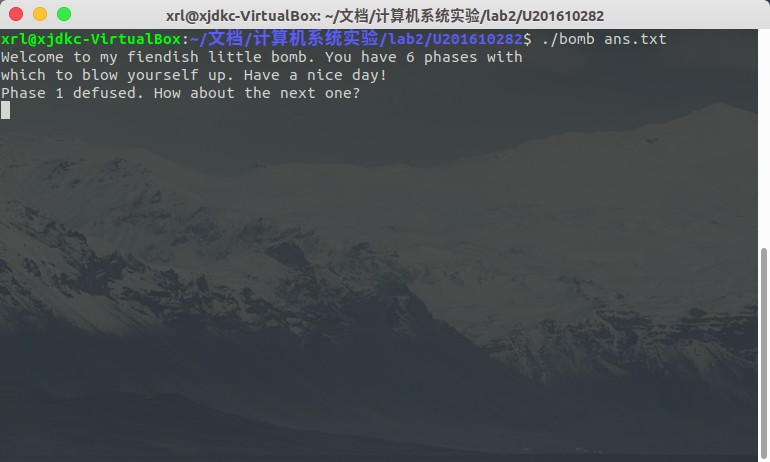


图2.2.1-4 phase\_1 破解图

* + 1. 阶段2: 循环

1. 任务描述

本阶段主要考察的是循环，需要通过反汇编工具objdump完成对于程序的反汇编并根据反汇编代码来完成，这一步在之前的准备工作中已经做了，下面的任务就是分析反汇编代码，找到循环部分并分析其内在逻辑得到数列递推公式和数列初值，将正确的输入字符串找出来。

1. 实验设计

本阶段实验需要找到正确的数字序列并将之输入，所以我们需要在反汇编代码中找到数组计算比较的循环部分，然后根据代码得到数组的递推公式，这样就可以得到正确的字符串了。本部分不需要用gdb

1. 实验过程

实验过程分为两步：

1. 查看并分析反汇编代码，获取数列递推公式和初值

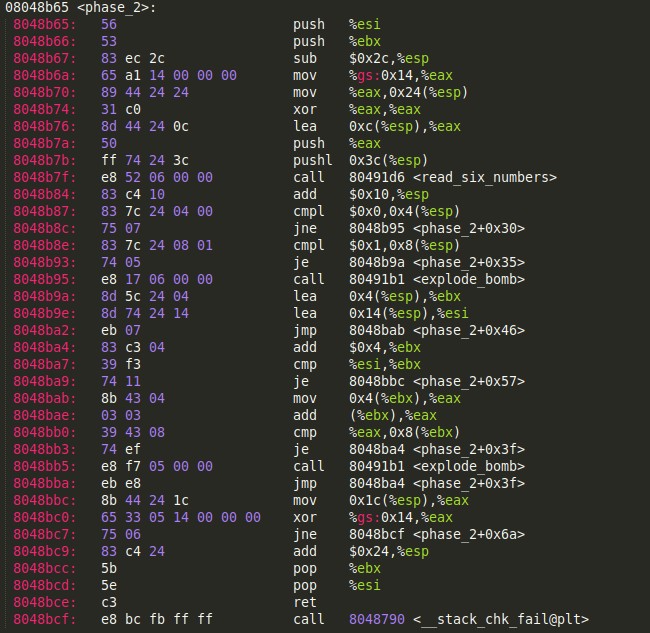


图2.2.2-1 phase\_2反汇编代码

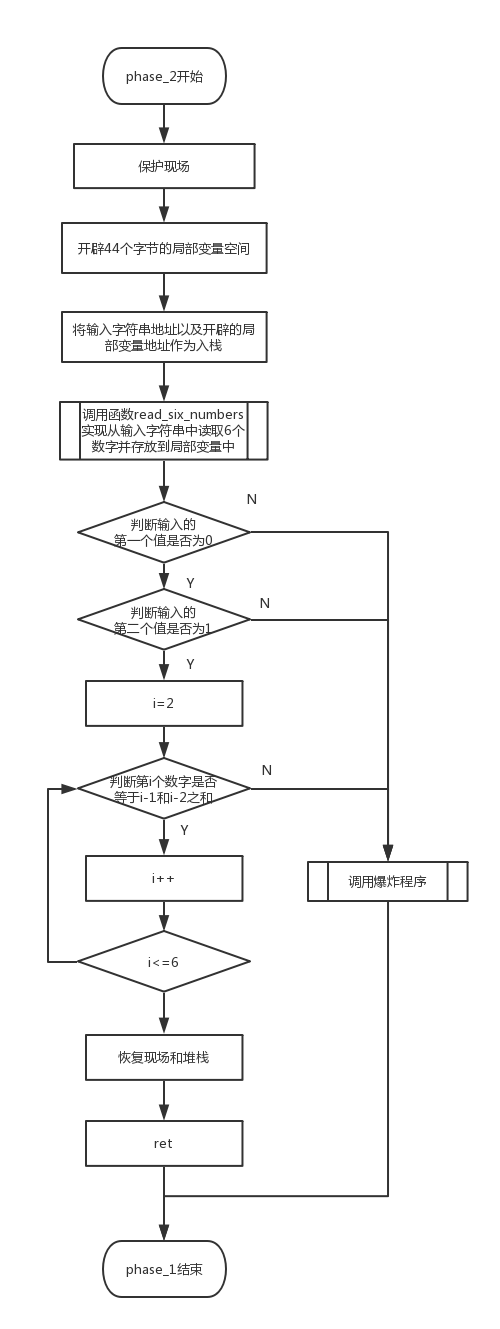


图2.2.2-2 phase\_2流程图

从中可以得知数列的初值为0和1即a1=0,a2=1

数列的递推公式为an=an-1+an-2

1. 根据公式获得字符串

即0 1 1 2 3 5

将得到字符串写入结果文件。

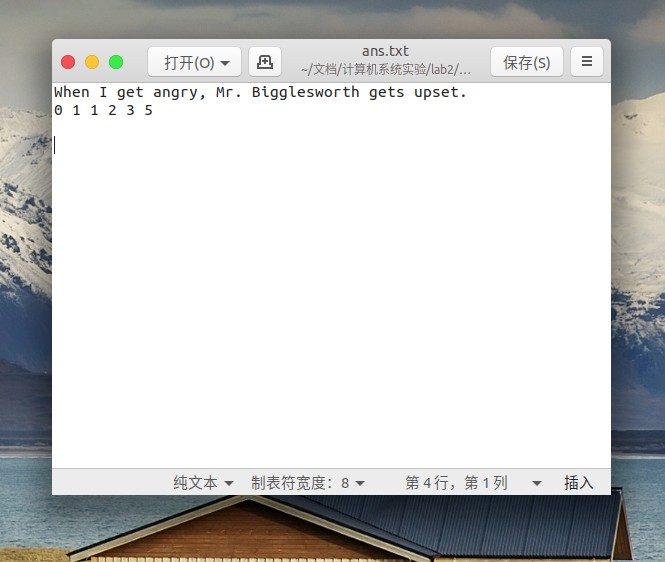


图2.2.2-3 phase\_2结束后破解字符串

1. 实验结果

在终端中将结果文件作为输入运行炸弹程序，成功破解！

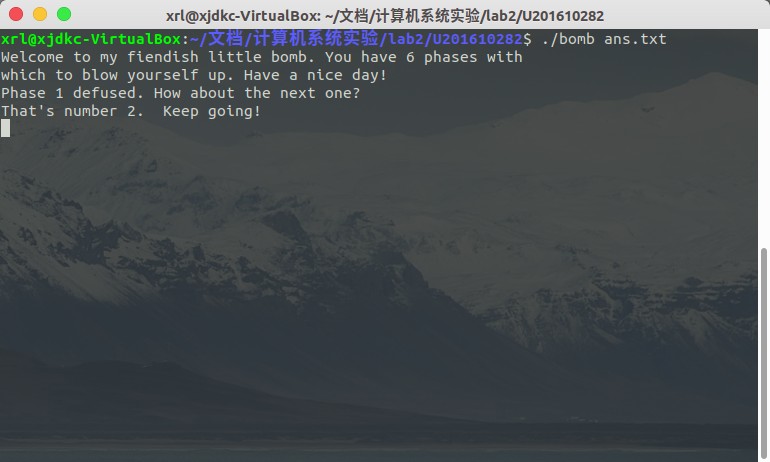


图2.2.2-4 phase\_2 破解图

* + 1. 阶段3: 条件/分支

1. 任务描述

本阶段主要考察的是条件和分支，需要通过反汇编工具objdump完成对于程序的反汇编并根据反汇编代码来完成，这一步在之前的准备工作中已经做了，下面的任务就是分析反汇编代码，找到分支部分然后根据判断条件获知正确字符串，实现对于炸弹的破解。

1. 实验设计

本阶段实验需要得到正确的字符串并将之输入，所以我们需要在反汇编代码中找到分支部分，每一个分支的判断条件来获得正确的字符串，这样就可以完成对于炸弹的破解了。本实验涉及了分支程序的地址表，需要使用gdb来查看地址表以获知跳转地点。

1. 实验过程

实验过程主要分为三步：

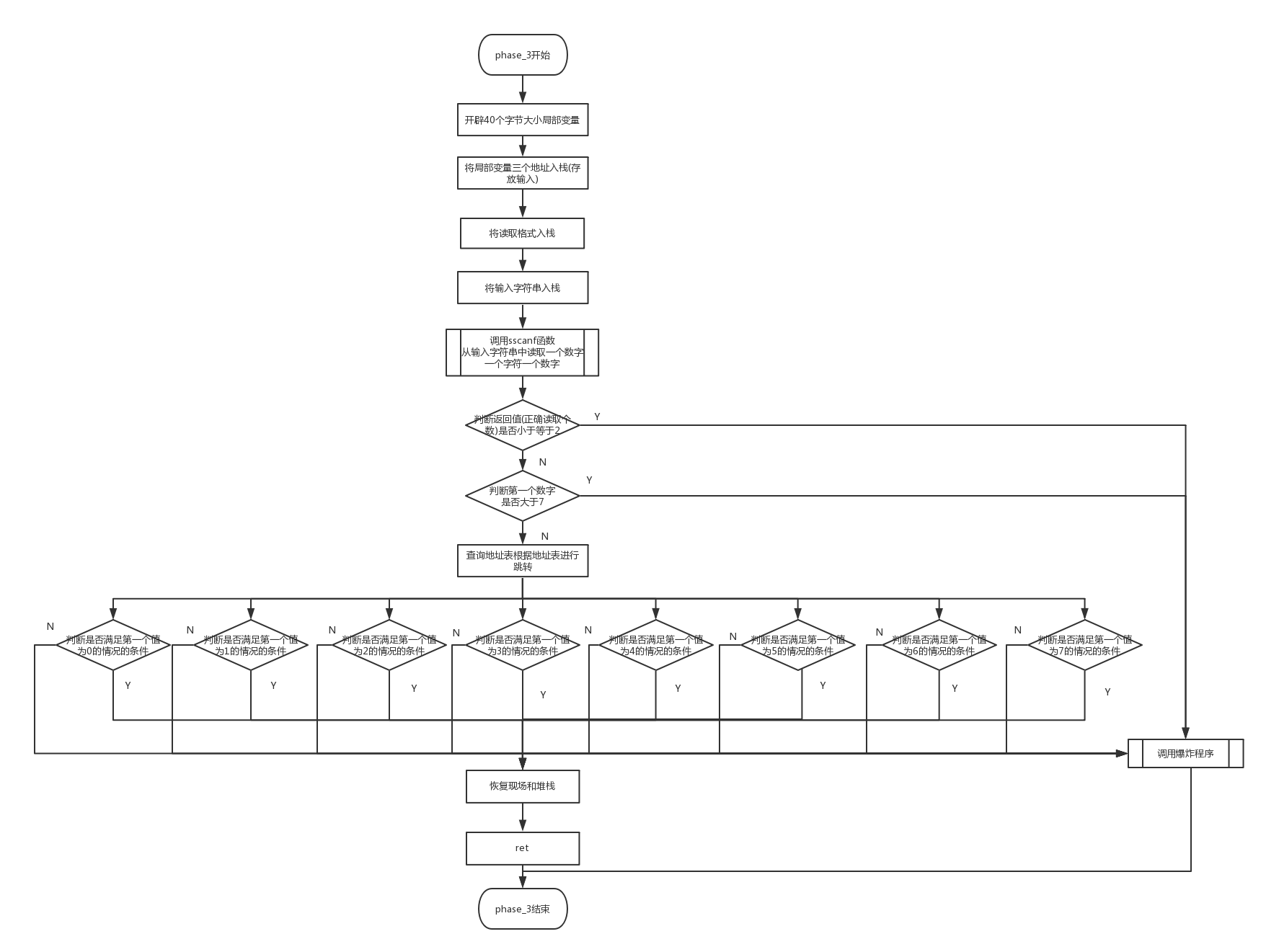
1. 查看并分析phase\_3反汇编代码，获知流程。

图2.2.3-1 phase\_3 流程图

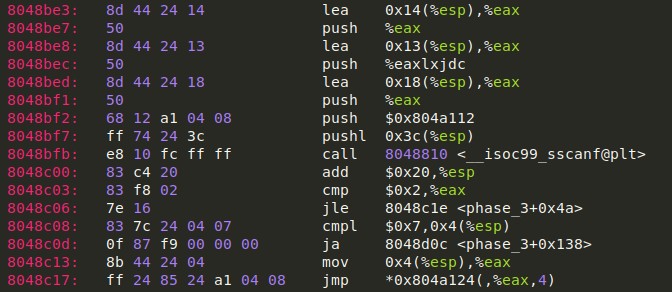


图2.2.3-2 phase\_3 部分反汇编代码图

可以从代码中知道跳转表地址在**0x804a124**处存放

1. 运用gdb查看地址表获知跳转地址

在gdb 中查看跳转表

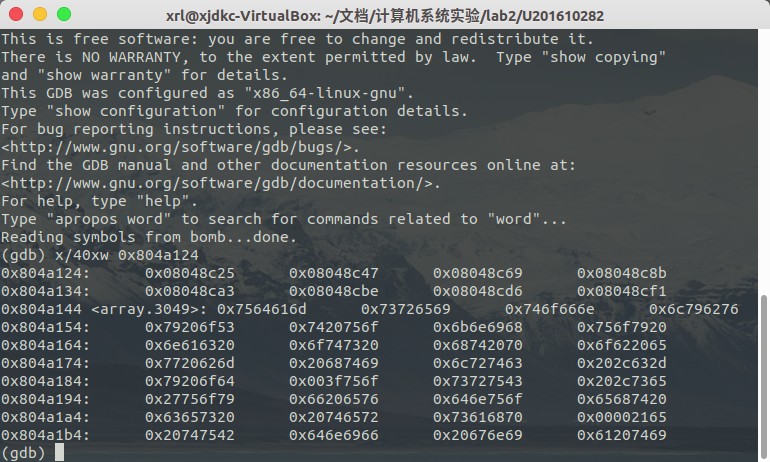


图2.2.3-3 phase\_3 gdb查看地址表图

可以发现phase\_3是根据第一个输入的数字来进行后面字符和数字的判断

1. 分析代码不同分支下的判断条件获取正确字符串

根据跳转表查看源代码的后续判断条件可以得到如下结果

0 a 635

1 x 210

2 h 370

3 z 105

4 m 886

5 f 101

6 y 610

7 c 603

将之任意一个作为phase\_3的输入字符串就可以破解phase\_3

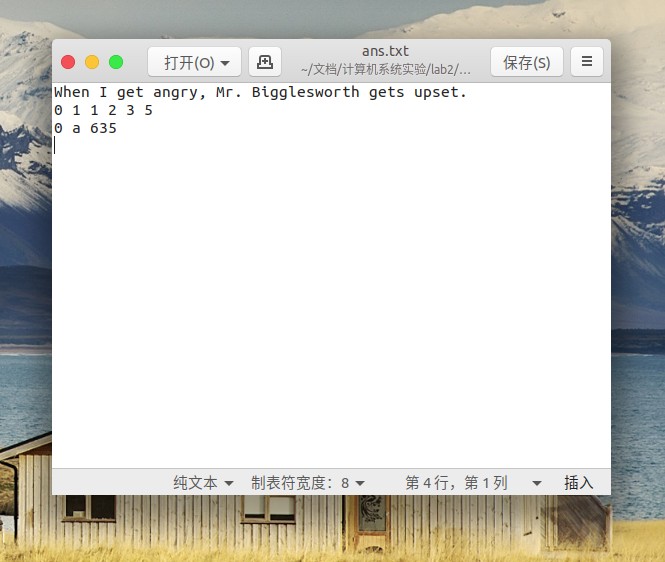


图2.2.3-4 phase\_3 结束后结果文件图

1. 实验结果

在终端中将结果文件作为输入运行炸弹程序，成功破解！

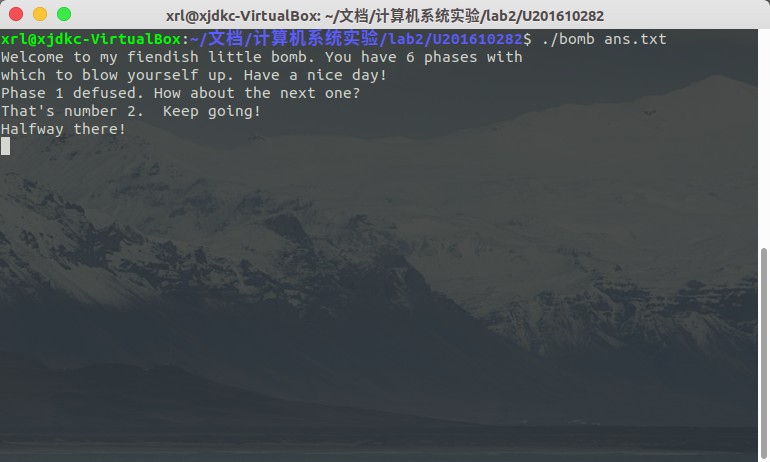


图2.2.2-4 phase\_2 破解图

* + 1. 阶段4: 递归调用和栈

1. 任务描述

本阶段主要考察的是递归调用和栈，需要通过反汇编工具objdump完成对于程序的反汇编并根据反汇编代码来完成，这一步在之前的准备工作中已经做了，下面的任务就是分析反汇编代码，分析函数调用时栈的变化调用，然后分析递归函数根据递归函数功能和phase\_4的程序功能反推字符串。

1. 实验设计

本阶段实验需要正确获知递归函数的功能，所以我们需要在反汇编代码中找到递归函数部分，然后根据反汇编代码推知递归函数功能，了解递归函数功能后回到phase\_4函数根据条件判断反推出字符串输入。

1. 实验过程

本阶段实验主要分为三个部分：

1. 分析phase\_4反汇编代码，读懂“炸弹”判断流程。

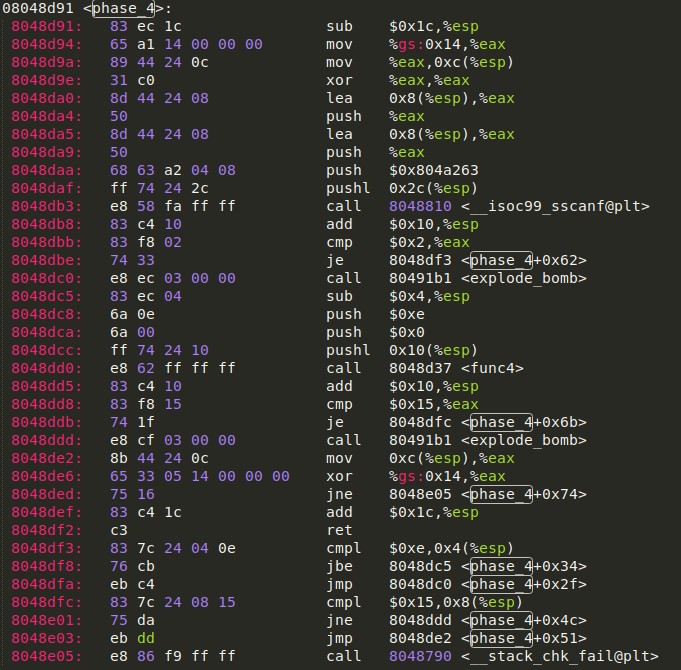


图2.2.4-1 phase\_4 反汇编代码

从phase\_4反汇编代码中我们可以得到阶段四判定的大致流程。

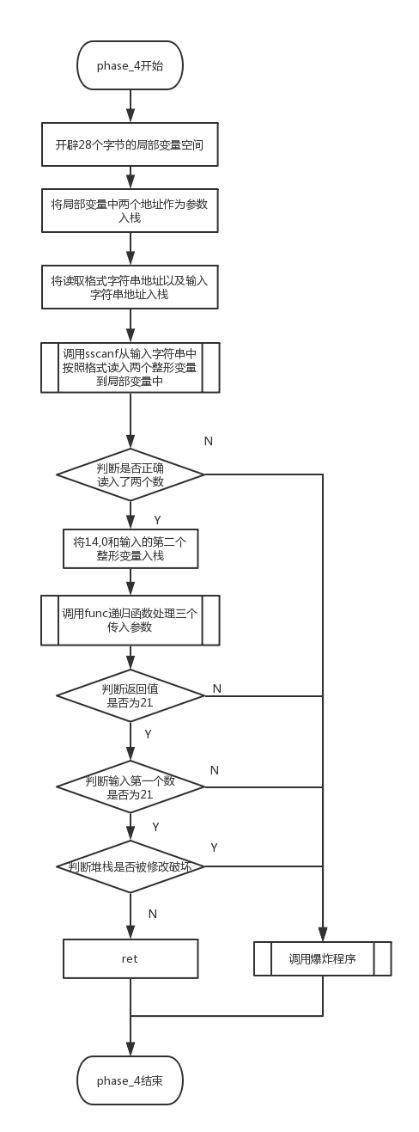


图2.2.4-2 phase\_4 流程图

其中我们可以通过gdb查看读入数据格式(查看0x8041263处字符串)



图2.2.4-3 phase\_4 数据读入格式

可以看到读入格式为两个整形，所以我们输入的字符串也将包含两个整形数字。

1. 分析func4递归函数，了解递归函数功能。

在1中我们已经知道我们需要输入特定的第二个整形数字以使得递归函数返回值为21。所以我们需要分析递归函数。

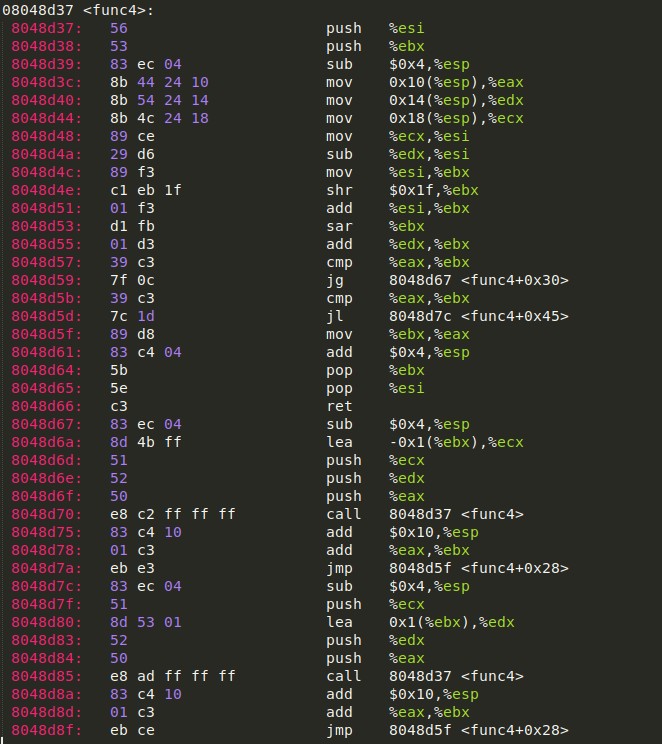


图2.2.4-4 func4递归函数反汇编代码

分析递归函数的反汇编代码我们可以得到递归函数的流程图。

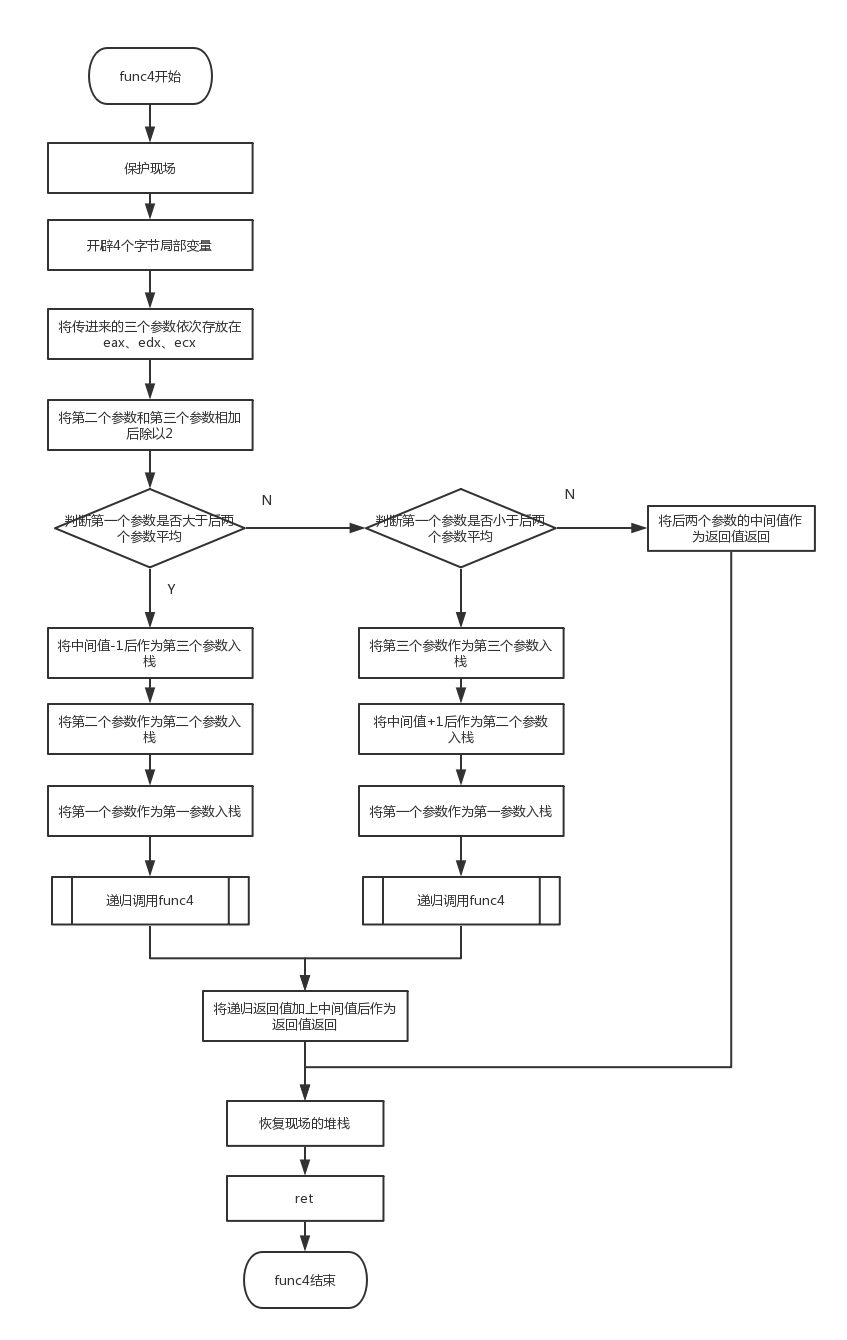


图2.2.4-5 func4递归函数流程图

由此我们可以分析得出，递归函数其实是一个二分查找的函数，它的功能是在第二个参数(区间下界)和第三个参数(区间上界)之间查找第一个参数，若区间中间值大于目标值则在前半区间查找，否则在后边区间查找，知道区间中间值为目标值，递归函数的返回值是每次查找区间中间值之和。将这个和作为返回值返回。

1. 根据递归函数func4的功能以及phase\_4判定流程反推出正确字符串。

我们在phase\_4的流程图中发现，我们需要让递归函数返回21，而递归函数是在0-14区间内查找我们输入的第一个数字，根据区间和递归函数的功能，我们可以得到如下二叉树。

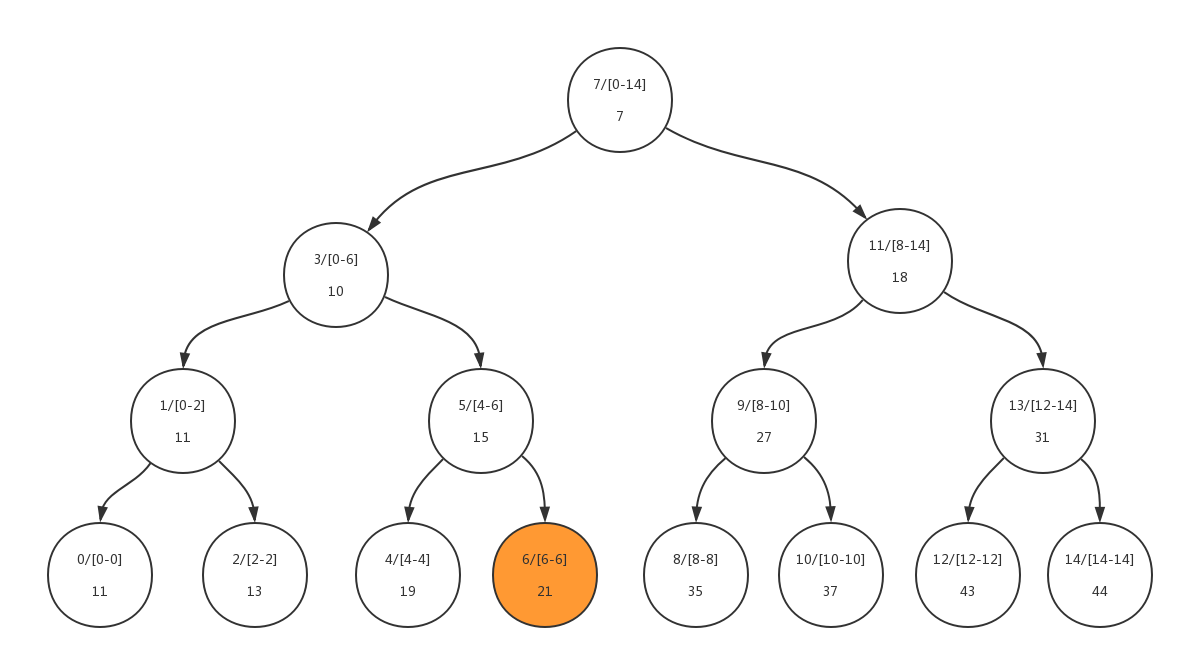


图2.2.4-6 func4递归函数各输入值流程图

这个图中每个节点都表示一次递归，第一个值为本次递归中区间的中间值，斜杠后面是区间返回，下面是进行到这次递归最终的返回值，由于返回值为21，所以即为图中橙色节点对应返回值，那么也就是说我们查找的结果为6。6即为我们要输入的第一个值。第二个数字值为21，故最终字符串就是“6 21”

1. 实验结果

我们将所得结果写入结果文件:

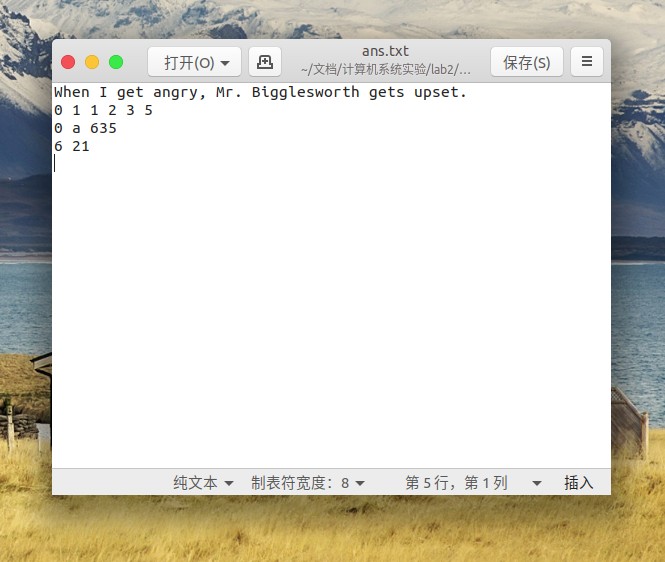


图2.2.4-7 phase\_4结束后结果文件图

在终端中将结果文件作为输入运行炸弹程序，成功破解！

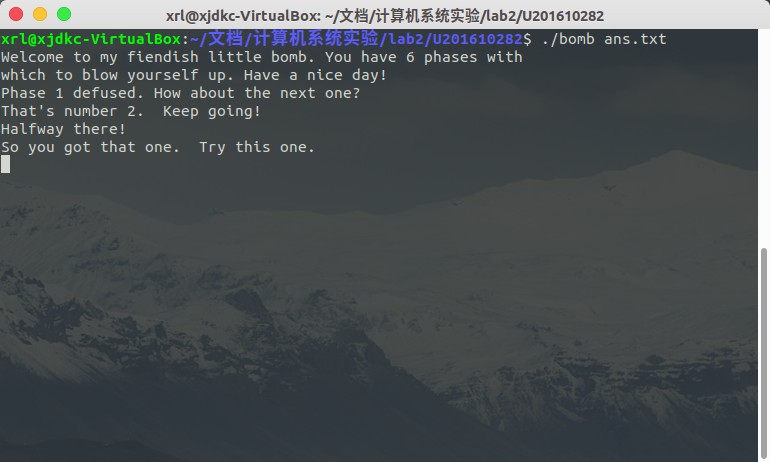


图2.2.4-8 phase\_4破解状态图

* + 1. 阶段5: 指针

1. 任务描述

本阶段主要考察的是指针，需要通过反汇编工具objdump完成对于程序的反汇编并根据反汇编代码来完成，这一步在之前的准备工作中已经做了，下面我们只需要分析阶段5是怎样通过输入来进行指针操作并判断输入字符串是否正确的，从而反推正确的字符串。

1. 实验设计

本阶段实验需要正确弄清楚phase\_5的操作流程，所以我们需要在反汇编代码中找到phase\_5部分，然后根据反汇编代码推知phase\_5判定流程，并通过gdb调试工具在内存中查看具体变量的值，因为在这其中需要用内存中变量来组成一个正确字符串。弄清流程我们就可以反推出输入字符串了。

1. 实验过程

本阶段实验分为三个部分:

1. 分析phase\_5反汇编代码，读懂“炸弹”判断流程。

首先我们需要分析phase\_5反汇编代码

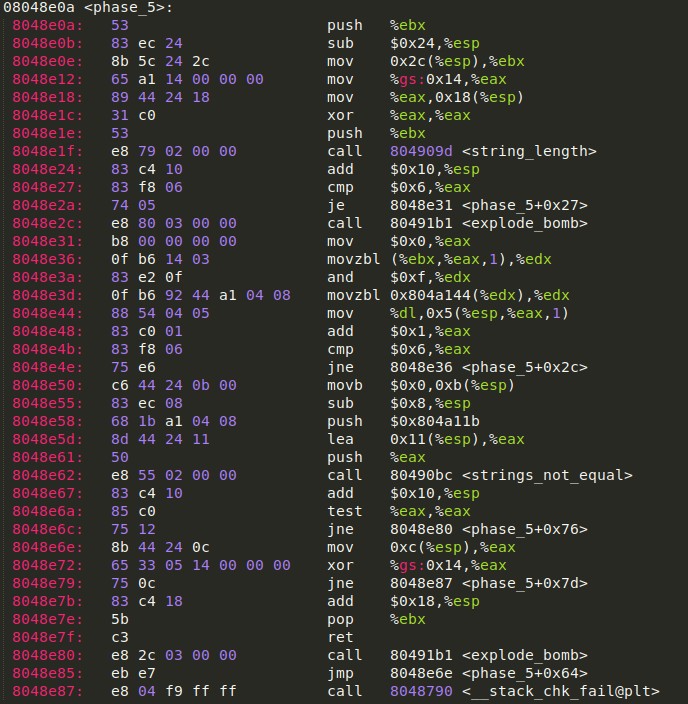


图2.2.5-1 phase\_5反汇编代码

由此我们可以得到phase\_5的流程图

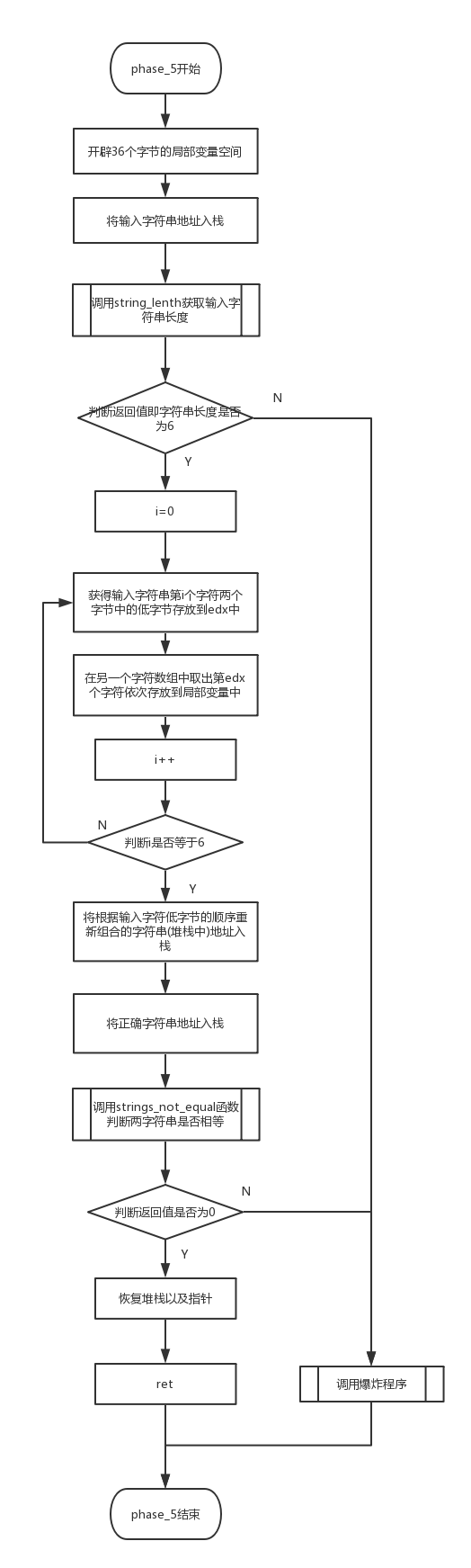


图2.2.5-2 phase\_5流程图

1. 使用gdb工具查看内存中字符，并查看正确字符串。

由1可知最终和正确字符串比较的字符串是根据我们输入的字符以及内存中的字符数组重新组合成的字符串进行比较的。

那么我们首先需要知道字符数组的内容(由于是根据字节的低位进行取字符的，4位二进制数只有0-15，故字符数组只取前16个)

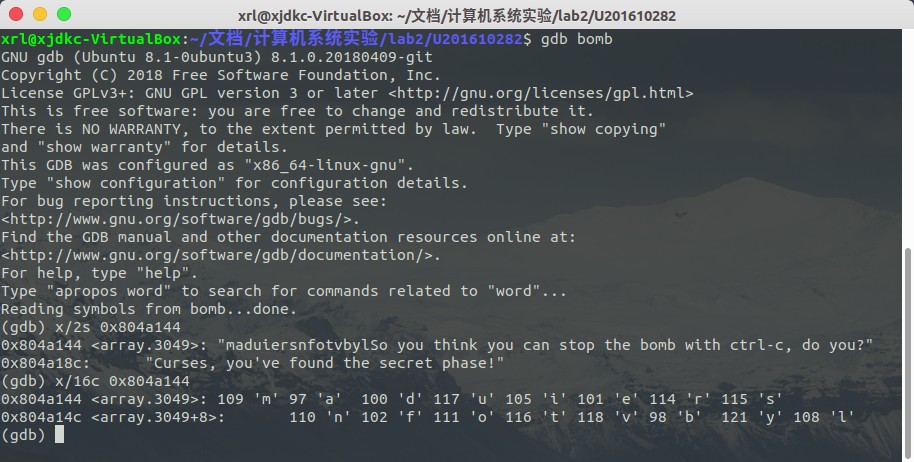


图2.2.5-3 gdb查看字符数组

由此可知字符数组内容从0-15分别为“madui ersnf otvby l”。

正确字符串为:



图2.2.5-4 gdb查看正确字符串

可知正确字符串为“flyers”。

1. 根据2中查看情况重组内存中字符以得到正确字符串，推出正确输入。

由2可知我们需要得到flyers，其中各个字符在字符数组中的下标为:

f 9

l 15

y 14

e 5

r 6

s 7

也就是说我们只要保证输入字符串长度为6并且每个字符的低位符合上面的数值即可成功破解，不妨用小写字母来破解(因为小写字母的低四位即小写字母的顺序，a的ascll码为60)。

利用小写字母来破解:

第一个字符为第9个小写字母 i

第二个字符为第15个小写字母 o

第三个字符为第14个小写字母 n

第四个字符为第5个小写字母 e

第五个字符为第6个小写字母 f

第六个字符为第7个小写字母 g

最终得到输入字符串为ionefg。

将输入字符串写入结果文件:



图2.2.5-5 phase\_5结束后结果文件状态图

1. 实验结果

在终端中将结果文件作为输入运行炸弹程序，成功破解！



图2.2.5-6 phase\_5破解图

* + 1. 阶段6: 链表/指针/结构

1. 任务描述

本阶段主要考察的是链表/指针/结构，需要通过反汇编工具objdump完成对于程序的反汇编并根据反汇编代码来完成，这一步在之前的准备工作中已经做了，下面我们只需要分析阶段6是怎样通过输入来进行链表/指针/结构操作并判断输入字符串是否正确的，从而反推正确的字符串。

1. 实验设计

本阶段实验需要正确弄清楚phase\_6的操作流程，所以我们需要在反汇编代码中找到phase\_6部分，然后根据反汇编代码推知phase\_6判定流程，并通过gdb调试工具在内存中查看具体变量的值。本阶段主要是需要通过gdb查看链表结果根据值以及phase\_6判定方式进行对链表的重新排序，在了解其判断流程以及值后即可获得正确字符串。

1. 实验过程

本次实验主要分为三步:

1. 分析phase\_6反汇编代码，读懂字符串以及指针、链表、结构操作流程。

首先分析phase\_6反汇编代码,本阶段代码有些长，主要分为三个部分:

1. 输入数字并判重

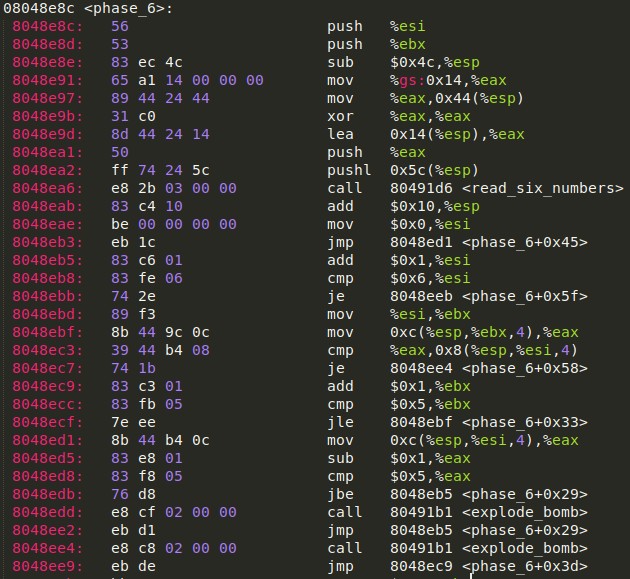


图2.2.6-1 phase\_6反汇编代码-输入数字并判重

1. 对链表进行重新排序

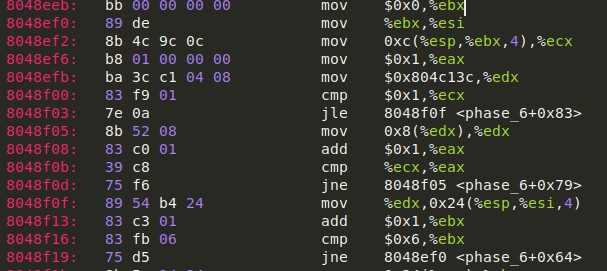


图2.2.6-2 phase\_6反汇编代码-链表重新排序

1. 判断升序并结束

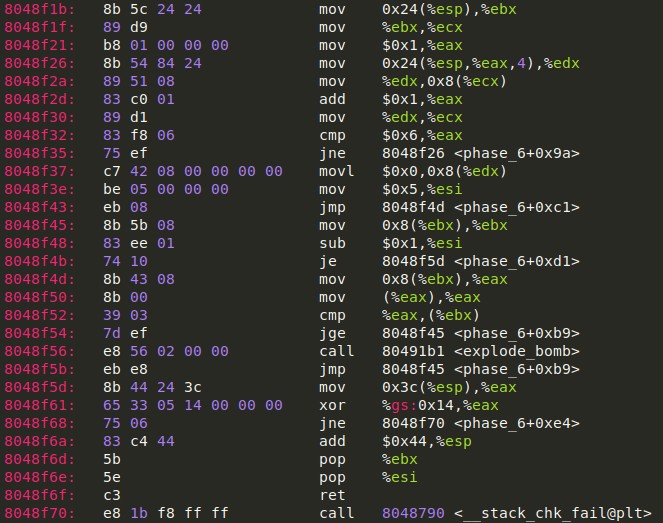


图2.2.6-3 phase\_6反汇编代码-判断升序

由此可以得到phase\_6的流程图:

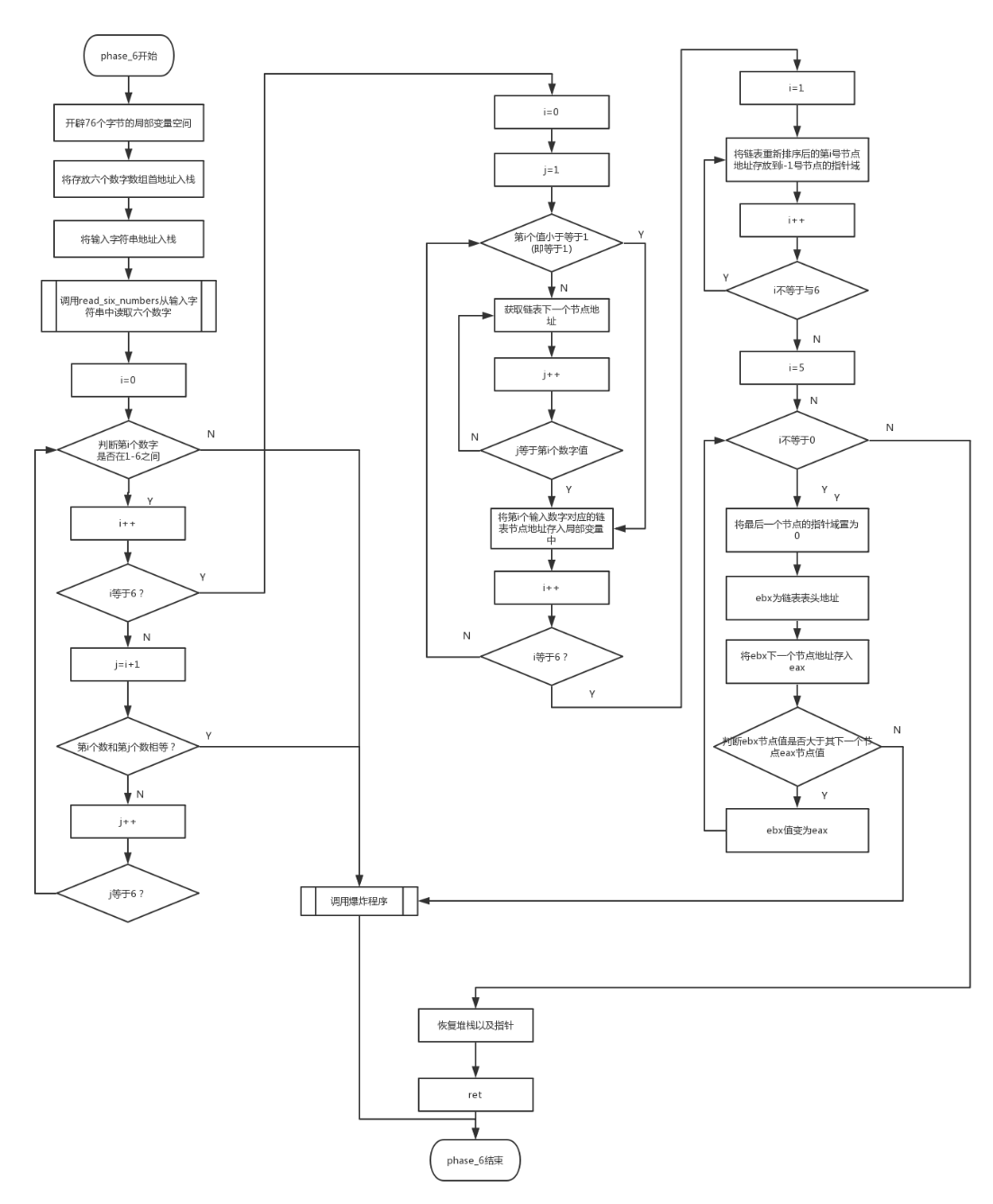


图2.2.6-3 phase\_6流程图

1. 利用gdb查看链表值的情况。

在程序中我们发现了链表以及结构体存放的地址0x804c13c，我们可以通过gdb来查看值以及结构体内的链表指针值。

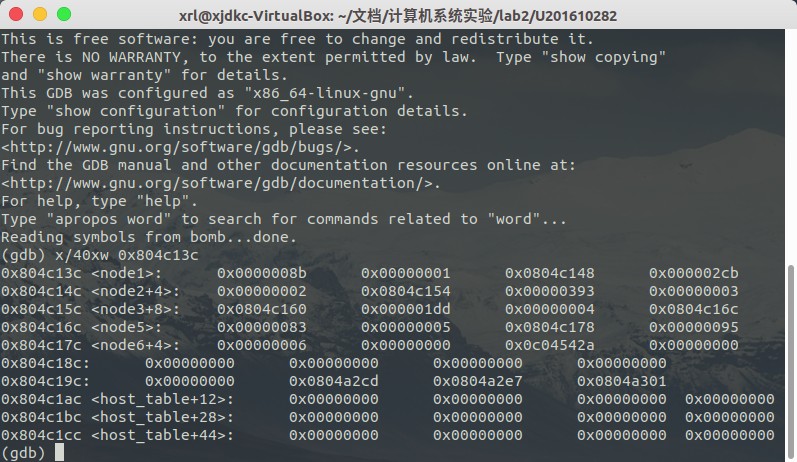


图2.2.6-4 phase\_6链表值

可以看到链表长度为6，包含六个结构体，每个结构体长度为3个4字节，第一个4字节表示该链表节点的值，第二个4字节表示该节点最初位为链表的第几个节点，第三个4字节表示下一个节点的地址。

由此我们可以得到如下结果

节点最初编号 值

1. 0x0000008b
2. 0x000002cb
3. 0x00000393
4. 0x000001dd
5. 0x00000083
6. 0x00000095
7. 根据1-2获得正确字符串。

由于根据之前分析，phase\_6根据我们的输入将链表进行重新排序，并且要保证链表以降序的形式进行排列，所以我们需要输入链表在最初的时候降序排序的序列，从上面得到的数据可以知道降序序列为

3 2 4 6 1 5

即最初第3个节点最大，第5个节点最小

将之作为字符串写入结果文件中:

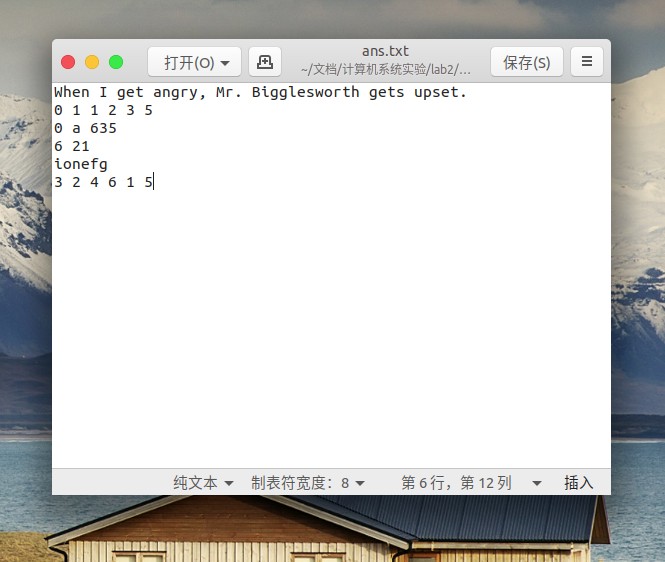


图2.2.6-5 phase\_6 结束后结果文件状态图

1. 实验结果

在终端中将结果文件作为输入运行炸弹程序，成功破解！



图2.2.6-5 phase\_6 破解状态图

* + 1. 阶段7: 隐藏阶段

1. 任务描述

本阶段需要我们在炸弹程序中找到进入隐藏阶段的方法，然后在隐藏阶段再输入一段字符串破解炸弹隐藏阶段。需要通过反汇编工具objdump完成对于程序的反汇编并根据反汇编代码来完成，这一步在之前的准备工作中已经做了，下面需要做的就是找到隐藏阶段的进入方法以及破解隐藏阶段。

1. 实验设计

本阶段需要弄清楚炸弹程序是怎样判断是否启动隐藏阶段，所以我们需要在炸弹程序中找到隐藏阶段的入口，根据提示炸弹程序是判断phase\_4阶段后面是否跟了一段特定的字符串从而判断是否进入隐藏阶段的，所以我们需要在找到隐藏阶段的入口的同时，找到启动隐藏阶段的字符串。

当进入隐藏阶段后，我们还需要破解隐藏阶段的炸弹，也就是说我们还要输入一段破解隐藏阶段的字符串也就是第七个字符串，在隐藏阶段的入口出当判断进入隐藏阶段后会调用隐藏阶段炸弹爆炸的判定函数，按照之前的方式分析该部分函数即可破解隐藏阶段。

1. 实验过程

本部分的实验过程主要分为6步:

1. 分析炸弹反汇编代码，找到隐藏阶段入口。

分析炸弹反汇编代码，发现在phase\_defused函数下有secret\_phase的入口，进一步分析phase\_defused函数反汇编代码。

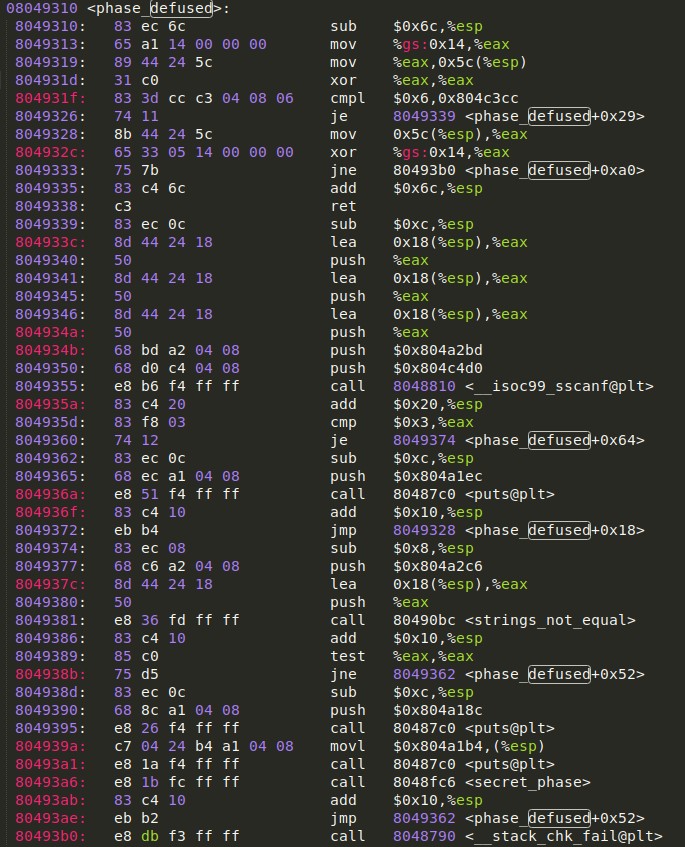


图2.2.7-1 phase\_defused 反汇编代码

从中可以发现，phase\_defused在读取六行数据之后进行了与之前不同了流程，先是从0x804c4d0这个字符串中按照0x804a2bd格式读取了三个数据(如果没有就直接按照正常流程结束)，然后对第三个数据代表的字符串进行了判断如果和另一个字符串相等就输出一些信息之后调用secret\_phase.猜测这里就是secret\_phase的入口。

1. 分析判定进入隐藏阶段入口找到引动字符串。

将1中的一些地址全部用gdb查看可以得到以下状态图:



图2.2.7-2 phase\_defused 一些字符串

结合1可以知道在phase\_defused中当前六个阶段都没有爆炸的前提下，炸弹重新对第四阶段的字符串进行了重新读入，读入方式为两个整型，一个字符串，而第四阶段读入格式为两个整型多出来的字符串一定就是隐藏阶段入口字符串，根据比较查看得到隐藏阶段入口字符串为“DrEvil”

1. 进入隐藏阶段。

将DrEvil附在第四阶段字符串后:



图2.2.7-3 开启隐藏阶段字符串

运行炸弹:

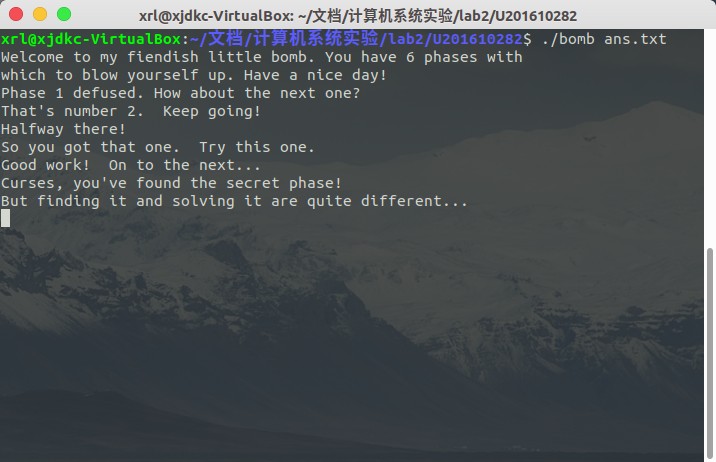


图2.2.7-4 开启隐藏阶段状态图

1. 分析隐藏阶段炸弹反汇编代码，了解判定爆炸条件以及流程。

经过以上三个步骤，终于进入了隐藏阶段，所以要怎么破解隐藏阶段呢?

首先我们还是要先分析隐藏阶段反汇编代码:

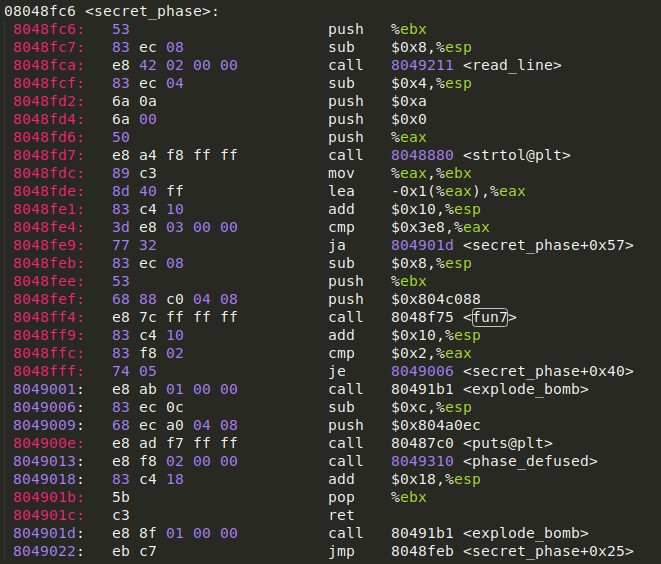


图2.2.7-5 secret\_phase 反汇编代码

由此可以得到隐藏阶段流程图

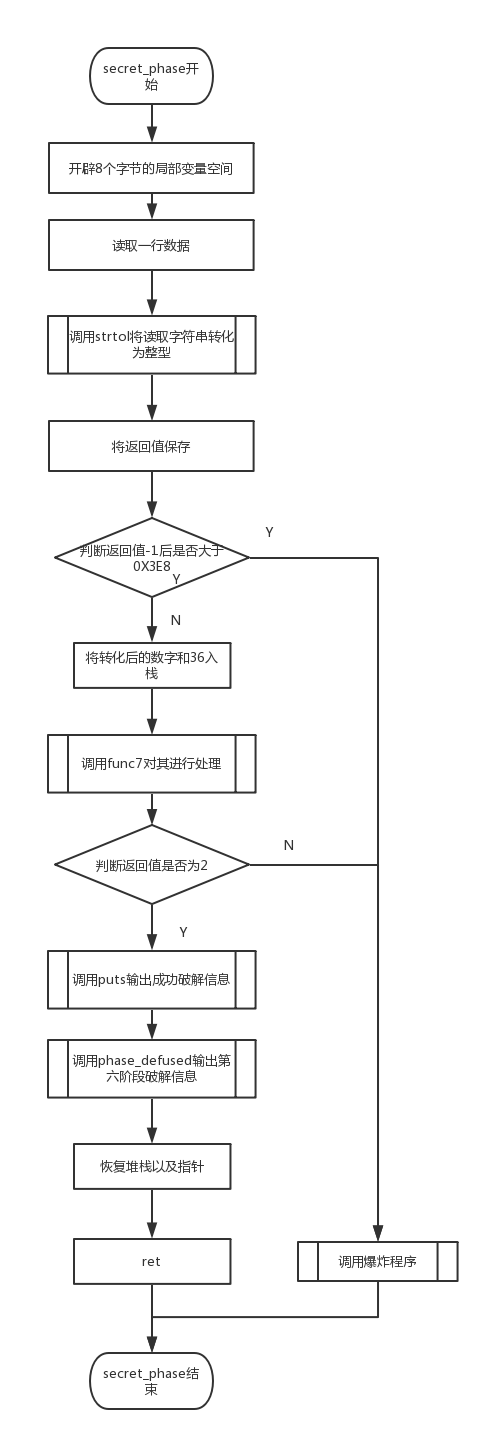


图2.2.7-6 secret\_phase 流程图

根据流程图可以知道，我们需要让在secret\_phase阶段输入的数字满足经过fun7处理好返回2，所以我们还要进一步了解fun7的功能。

1. 分析隐藏阶段fun7递归函数，了解递归函数功能

进一步分析fun7代码：

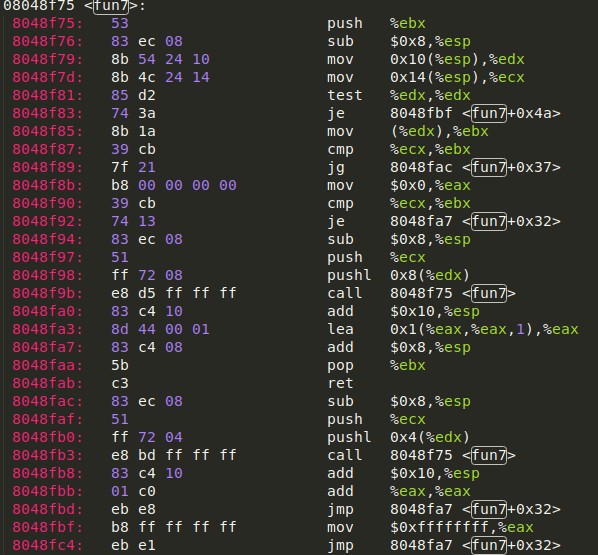


图2.2.7-7 fun7 反汇编代码

然后我们就可以根据反汇编代码分析出fun7的流程图:



图2.2.7-8 fun7 流程图

由此可以得知fun7的功能是判断输入数字是否等于输入地址存的值，若等于返回0，若大于则跳转到地址下一个变量存的地址进行判断，若小于则跳转到地址下下个变量存的地址进行判断，知道等于位置，返回值根据判断情况进行不断增加。

1. 根据4-5反推隐藏阶段正确字符串破解隐藏阶段炸弹。

由于要得到2，所以之前肯定经历了一次乘2+1和依次乘2的过程，也就是说在隐藏阶段fun7递归了三层，在第三层的时候发现相等返回0，然后在第二层的时候发现小于计算0\*2+1=1将1返回，然后在第一层发现大于，将1\*2后返回2得到正确结果。

首先用gdb查看初始地址:

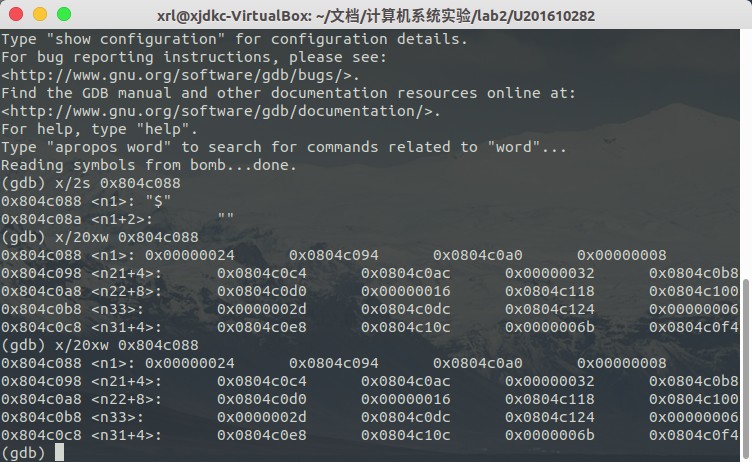


图2.2.7-8 判断大小数值以及转移地址

第一层递归:

第一个值为0x24 层递归按照之前推理判断结果应为大于输入值，即跳转到0x0804c094进行判断。

第二层递归:

对0x0804c094存放的值进行判断，存放的值为0x8，此层递归按照之前推理判断结果应为小于输入值，即跳转到0x0804c0ac进行判断。

第三层递归:

对0x0804c0ac进行判断，此层递归按照之前推理应该是等于输入值，得到输入值为0x16=22。故输入值为22。

将22写入结果文件:



图2.2.7-9 secret\_phase 结束时结果文件状态图

1. 实验结果

在终端中将结果文件作为输入运行炸弹程序，成功破解！

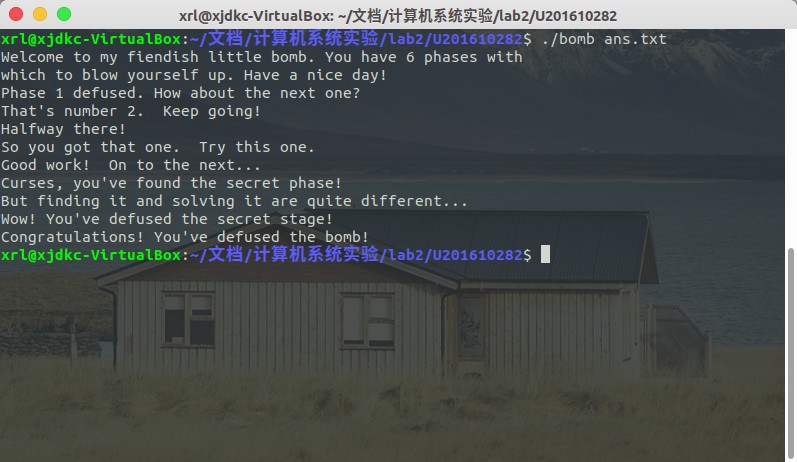


图2.2.7-10 secret\_phase 破解状态图

* 1. 实验小结

本次实验在Linux环境下通过objdump和gdb工具实现了对于“炸弹”程序的反汇编和调试。学习到了很多命令行的东西，以及在Linux环境下的一系列操作。

通过对于炸弹六个阶段的破解以及隐藏阶段的破解，加深巩固了对于字符串操作、循环、条件/分支、递归调用和栈、指针、链表/指针/结构的机器级实现了理解。特别是对于函数参数传递、堆栈的使用变化情况、指针的实现、链表结构的实现有了更为清晰的认识。

本次实验十分有趣，在拆除“炸弹”的过程中加深了对于程序运行的理解，完成的也比较顺利，不过对于一些细节的地方还要多加注意，这样会减少不必要的时间耽误。