華中科技大學

课 程 实 验 报 告

课程名称：计算机系统基础

专 业 班 级：	计卓 1501
学　　号：	U201514898
姓　　名：	胡思勖
指 导 教 师：	谭志虎
报 告 日 期：	2017 年 6 月 1 日

目录

I 实验概述 **1**

I.1 实验目标、要求 1

I.2 实验意义 1

II 内容 **2**

II.1 阶段 1 2

II.1.1 任务描述 2

II.1.2 实验过程 2

II.1.3 实验结果 3

II.2 阶段 2 3

II.2.1 任务描述 3

II.2.2 实验过程 3

II.2.3 实验结果 4

II.3 阶段 3 4

II.3.1 任务描述 4

II.3.2 实验过程 5

II.3.3 实验结果 8

II.4 阶段 4 8

II.4.1 任务描述 8

II.4.2 实验过程 8

II.4.3 实验结果 11

II.5 阶段 5 11

II.5.1 任务描述 11

II.5.2 实验过程 11

II.5.3 实验结果 13

II.6 阶段 6 13

II.6.1 任务描述 13

II.6.2 实验过程 13

II.6.3 实验结果 15

II.7 隐藏阶段 15

II.7.1 任务描述 15

II.7.2 实验过程 16

II.7.3 实验结果 20

III 实验小结 **20**

I 实验概述

I.1 实验目标、要求

本实验中, 你要使用课程所学知识拆除一个“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。六个层次如下:

- 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
- 隐藏阶段: 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。

I.2 实验意义

本实验用于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。

II 内容

II.1 阶段 1

II.1.1 任务描述

第一阶段主要检测反汇编中的基础部分，通过对于字符串比较过程的解析来拆除炸弹的第一阶段。

II.1.2 实验过程

- 首先，对于程序进行反汇编，观察主程序入口的反汇编结果，发现第一阶段调用的为 phase_1 函数。此外，还发现第 n 个阶段调用的为 phase_<n> 函数。在每个函数调用完毕后，调用 phase_defused 函数。由此可以推测，第一阶段的主要任务是分析 phase_1 的内容。

- 跳转到 phase_1 的部分，发现其反汇编代码如下：

```
1  08048b90 <phase_1>:
2   8048b90:   83 ec 1c                sub    $0x1c,%esp
3   8048b93:   c7 44 24 04 24 a3 04    movl   $0x804a324,0x4(%esp)
4   8048b9a:   08
5   8048b9b:   8b 44 24 20             mov    0x20(%esp),%eax
6   8048b9f:   89 04 24                mov    %eax,(%esp)
7   8048ba2:   e8 a3 04 00 00          call   804904a <strings_not_equal>
8   8048ba7:   85 c0                   test   %eax,%eax
9   8048ba9:   74 05                   je     8048bb0 <phase_1+0x20>
10  8048bab:   e8 15 07 00 00          call   80492c5 <explode_bomb>
11  8048bb0:   83 c4 1c                add    $0x1c,%esp
12  8048bb3:   c3                      ret
```

- 发现此函数调用了 strings_not_equal 子过程。而为 string_not_equal 子过程传入的参数一个为 phase_1 的传如参数，另一为常量地址 0x804a324。

- 根据函数名称和传入参数可以推测，此函数即为将传入字符串与一固定字符串进行比较，且 0x804a324 处即为此常量字符串。

- 对于 bomb 二进制文件从 0x804a324 处进行反汇编，如图2.1.1所示。

```
~/t/c/h/bomb807 objdump --start-address 0x804a324 -s bomb | head

bomb:     file format elf32-i386

Contents of section .rodata:
 804a324 4920616d 206e6f74 20706172 74206f66  I am not part of
 804a334 20746865 2070726f 626c656d 2e204920   the problem. I
 804a344 616d2061 20526570 75626c69 63616e2e  am a Republican.
 804a354 00000000 576f7721 20596f75 27766520  ....Wow! You've
 804a364 64656675 73656420 74686520 73656372  defused the secr
 804a374 65742073 74616765 21000000 4c8c0408  et stage!...L...
```

图 2.1.1: bomb 从 0x804a324 的反汇编内容

- 由此可以看出，此固定字符串的内容为 “I am not part of the problem. I am a Republican.”
- 将此字符串输入 bomb 程序，第一阶段成功解除，如图2.1.2所示。

```
~/t/c/h/bomb807  ./bomb                          Fri 02 Jun 2017 07:58:57 PM CST
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
```

图 2.1.2: 第一阶段解除

II.1.3 实验结果

由实验过程可知，第一阶段的解为字符串 “I am not part of the problem. I am a Republican.”，第一阶段成果解除。

II.2 阶段 2

II.2.1 任务描述

阶段二需要我们通过对于汇编中循环的分析来解析程序，从而推测这一阶段的解。

II.2.2 实验过程

- 由第一阶段可知，第二阶段应该从 phase_2 开始。
- 在反汇编中跳转到 phase_2 的部分，反汇编代码如下 (已添加必要注释):

```
08048bb4 <phase_2>:
 8048bb4:   56                      push   %esi
 8048bb5:   53                      push   %ebx
 8048bb6:   83 ec 34                sub    $0x34,%esp

////// parameter
 8048bb9:   8d 44 24 18             lea    0x18(%esp),%eax
 8048bbd:   89 44 24 04             mov    %eax,0x4(%esp)

 8048bc1:   8b 44 24 40             mov    0x40(%esp),%eax
 8048bc5:   89 04 24                mov    %eax,(%esp)
//////
 8048bc8:   e8 37 07 00 00          call   8049304 <read_six_numbers>
 8048bcd:   83 7c 24 18 01          cmpl   $0x1,0x18(%esp)
 8048bd2:   74 1e                   je     8048bf2 <phase_2+0x3e> //je L1

 8048bd4:   e8 ec 06 00 00          call   80492c5 <explode_bomb>
 8048bd9:   eb 17                   jmp    8048bf2 <phase_2+0x3e> // jmp L1

LOOP:
 8048bdb:   8b 43 fc                mov    -0x4(%ebx),%eax
 8048bde:   01 c0                   add    %eax,%eax
 8048be0:   39 03                   cmp    %eax,(%ebx)
 8048be2:   74 05                   je     8048be9 <phase_2+0x35>
```

```
25   8048be4:   e8 dc 06 00 00          call   80492c5 <explode_bomb>
26   8048be9:   83 c3 04                add    $0x4,%ebx
27   8048bec:   39 f3                   cmp    %esi,%ebx
28   8048bee:   75 eb                   jne    8048bdb <phase_2+0x27> // LOOP
29   8048bf0:   eb 0a                   jmp    8048bfc <phase_2+0x48> // LOOP end
30
31  L1:
32   8048bf2:   8d 5c 24 1c             lea    0x1c(%esp),%ebx
33   8048bf6:   8d 74 24 30             lea    0x30(%esp),%esi
34   8048bfa:   eb df                   jmp    8048bdb <phase_2+0x27> // LOOP
35
36   8048bfc:   83 c4 34                add    $0x34,%esp
37   8048bff:   5b                      pop    %ebx
38   8048c00:   5e                      pop    %esi
39   8048c01:   c3                      ret
```

- 从代码中可以看出，在 0x8048bc8 处调用了 read_six_numbers 函数，从函数名推测，应该需要读入六个整数。其中 parameter 注释部分为 read_six_numbers 的实参。
- 在读入六个整数后，将第一个整数与 $0x1 进行比较，如果不相等则炸弹爆炸，因此推断出第一个整数为 1。
- 对于接下来的标签的分析，可以发现其为一个 do-while 型循环，初始化 esi 为读入六个字符数组的超尾地址，ebi 数组的第二个元素，每次 ebi 向后移动一个整型大小，直到循环完整个数组为止。
- 在循环体 (0x8048bdb - 8048be4) 中，从第二个元素开始，将每一个元素与前一个元素进行比较，如果不为前一个元素的两倍则炸弹爆炸，因此推测第二个炸弹的解为一个等比数列，且初始值为 1，因此解为 “1 2 4 8 16 32”。
- 输入炸弹，成功拆除第二层，如图2.2.1所示。

```
~/t/c/h/bomb807  ./bomb                          Fri 02 Jun 2017 08:36:23 PM CST
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am not part of the problem. I am a Republican.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
```

图 2.2.1: 第二阶段炸弹拆除

II.2.3 实验结果

第二阶段的解为字符串 “1 2 4 8 16 32”。

II.3 阶段 3

II.3.1 任务描述

阶段三考察对于条件与分支判断的汇编语言解析，需要通过解析汇编条件判断的结构来拆除第三层。

II.3.2 实验过程

- 在反汇编中跳转到 phase_3 的部分，反汇编代码如下 (已添加必要注释):

```
08048c02 <phase_3>:
 8048c02:   83 ec 2c                sub    $0x2c,%esp

// ===
 8048c05:   8d 44 24 1c             lea    0x1c(%esp),%eax          // a pointer on
     stack
 8048c09:   89 44 24 0c             mov    %eax,0xc(%esp)

 8048c0d:   8d 44 24 18             lea    0x18(%esp),%eax          // a pointer on
     stack
 8048c11:   89 44 24 08             mov    %eax,0x8(%esp)

 8048c15:   c7 44 24 04 b1 a5 04    movl   $0x804a5b1,0x4(%esp)
 8048c1d:   8b 44 24 30             mov    0x30(%esp),%eax          // input parameter

 8048c21:   89 04 24                mov    %eax,(%esp)
// ===
 8048c24:   e8 37 fc ff ff          call   8048860 <__isoc99_sscanf@plt>
 8048c29:   83 f8 01                cmp    $0x1,%eax
 8048c2c:   7f 05                   jg     8048c33 <phase_3+0x31>
 8048c2e:   e8 92 06 00 00          call   80492c5 <explode_bomb>

 8048c33:   83 7c 24 18 07          cmpl   $0x7,0x18(%esp)
 8048c38:   77 66                   ja     8048ca0 <phase_3+0x9e>    boom

 8048c3a:   8b 44 24 18             mov    0x18(%esp),%eax
 8048c3e:   ff 24 85 80 a3 04 08    jmp    *0x804a380(,%eax,4)
 8048c45:   b8 00 00 00 00          mov    $0x0,%eax
 8048c4a:   eb 05                   jmp    8048c51 <phase_3+0x4f>
 8048c4c:   b8 c1 03 00 00          mov    $0x3c1,%eax
 8048c51:   2d 3c 02 00 00          sub    $0x23c,%eax
 8048c56:   eb 05                   jmp    8048c5d <phase_3+0x5b>
 8048c58:   b8 00 00 00 00          mov    $0x0,%eax
 8048c5d:   05 a4 01 00 00          add    $0x1a4,%eax
 8048c62:   eb 05                   jmp    8048c69 <phase_3+0x67>
 8048c64:   b8 00 00 00 00          mov    $0x0,%eax
 8048c69:   2d 41 01 00 00          sub    $0x141,%eax
 8048c6e:   eb 05                   jmp    8048c75 <phase_3+0x73>
 8048c70:   b8 00 00 00 00          mov    $0x0,%eax
 8048c75:   05 41 01 00 00          add    $0x141,%eax
 8048c7a:   eb 05                   jmp    8048c81 <phase_3+0x7f>
 8048c7c:   b8 00 00 00 00          mov    $0x0,%eax
 8048c81:   2d 41 01 00 00          sub    $0x141,%eax
 8048c86:   eb 05                   jmp    8048c8d <phase_3+0x8b>
 8048c88:   b8 00 00 00 00          mov    $0x0,%eax
 8048c8d:   05 41 01 00 00          add    $0x141,%eax
 8048c92:   eb 05                   jmp    8048c99 <phase_3+0x97>
 8048c94:   b8 00 00 00 00          mov    $0x0,%eax
 8048c99:   2d 41 01 00 00          sub    $0x141,%eax
 8048c9e:   eb 0a                   jmp    8048caa <phase_3+0xa8>
```

- 可以看出，首先调用了 sscanf 函数，且 sscanf 函数的第二个参数 (格式化字符串) 为一

常量字符串，且存储在 0x804a5b1 处。通过从这一地址开始反汇编，可以得到格式化字符串为 “%d %d”，如图2.3.1所示。

```
~/t/c/h/bomb807 objdump --start-address 0x804a5b1 -s bomb | head

bomb:     file format elf32-i386

Contents of section .rodata:
 804a5b1 256420 25640045 72726f72 3a205072 65 %d %d.Error: Pre
 804a5c1 6d6174 75726520 454f4620 6f6e2073 74 mature EOF on st
 804a5d1 64696e 00475241 44455f42 4f4d4200 45 din.GRADE_BOMB.E
 804a5e1 72726f 723a2049 6e707574 206c696e 65 rror: Input line
 804a5f1 20746f 6f206c6f 6e67002a 2a2a7472 75  too long.***tru
 804a601 6e6361 7465642a 2a2a0025 64202564 20 ncated***.%d %d
```

图 2.3.1: 格式化字符串的反汇编

- 在反汇编之后，将读入的第一个数与 $0x7 比较 (0x8048c33 处)，若比 7 大则炸弹爆炸，说明第一个数比 7 小。
- 观察到后面的程序进行了一系列的跳转，较为难以进行静态分析，因此直接打开 GDB 调试。在命令行中输入 gdb bomb 后，在 0x8048c3a 处打上断点，将前两题的答案写入 result.txt 中，然后在 gdb 中输入 r result.txt;
- 进入第三阶段后，输入 “1 1”，然后单步执行观察程序运行状态。如图2.3.2所示。

```
~/t/c/h/bomb807  gdb bomb                    Sat 03 Jun 2017 12:48:47 PM CST
GNU gdb (GDB) 7.12.1
Copyright (C) 2017 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-pc-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...done.
>>> dashboard -output /dev/pts/3
>>> b *0x8048c3a
Breakpoint 1 at 0x8048c3a
>>> r result.txt
Starting program: /home/husixu/t/computerSystem/homework_2/bomb807/bomb result.
txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
1 1

Breakpoint 1, 0x08048c3a in phase_3 ()
>>> ni
0x08048c3e in phase_3 ()
>>> ni
0x08048c45 in phase_3 ()
>>> ni
0x08048c4a in phase_3 ()
>>> ni
0x08048c51 in phase_3 ()
>>> ni
0x08048c56 in phase_3 ()
>>>
```

图 2.3.2: 进入 gdb 调试

- 发现单步执行一段时间后，发现执行到了 0x8048caa 处，首先将 scanf 获得的第一个数字与 $0x5 比较，若大于 5 则跳转到炸弹爆炸位置，然后将 eax 中的内容与 scanf 获得的第二个数字比较，如果相等则结束程序，否则炸弹爆炸。而输入的第一个参数为 1，满足小于 5 的条件，因此只要保证运行到此处时第二个参数与%eax 中的内容相等即可。
- 根据调试信息，此时%eax 中的内容为 0xfffffe27，如图2.3.3所示，转化为 10 进制为-473。由此断定第二个数为-473。

```
── Assembly ──
0x08048ca5 phase_3+163 mov    $0x0,%eax
0x08048caa phase_3+168 cmpl   $0x5,0x18(%esp)
0x08048caf phase_3+173 jg     0x8048cb7 <phase_3+181>
0x08048cb1 phase_3+175 cmp    0x1c(%esp),%eax
0x08048cb5 phase_3+179 je     0x8048cbc <phase_3+186>
0x08048cb7 phase_3+181 call   0x80492c5 <explode_bomb>
0x08048cbc phase_3+186 add    $0x2c,%esp
── Expressions ──
── History ──
── Memory ──
── Registers ──
   eax 0xfffffe27          ecx 0x00000000          edx 0xffffd21c
   ebx 0xffffd2e4          esp 0xffffd200          ebp 0xffffd248
   esi 0xf7f94000          edi 0x00000000          eip 0x08048cb1
eflags [ CF PF AF SF IF ]   cs 0x00000023          ss 0x0000002b
    ds 0x0000002b           es 0x0000002b          fs 0x00000000
    gs 0x00000063
── Source ──
── Stack ──
[0] from 0x08048cb1 in phase_3+175
(no arguments)
[1] from 0x08048b18 in main+251 at bomb.c:89
arg argc = 2
arg argv = 0xffffd2e4
── Threads ──
```

图 2.3.3: %eax 中的内容

- 退出 gdb，将第阶段的结果写入 result.txt 执行 bomb result.txt，第阶段成功解除。如图2.3.4所示。

```
~/t/c/h/bomb807 cat result.txt                Sat 03 Jun 2017 01:08:25 PM CST
I am not part of the problem. I am a Republican.
1 2 4 8 16 32
1 -473
~/t/c/h/bomb807 ./bomb result.txt             Sat 03 Jun 2017 01:08:26 PM CST
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
```

图 2.3.4: 第三阶段解除

II.3.3 实验结果

第三阶段的解为字符串 “1 2 4 8 16 32”，第三阶段成功拆除。

II.4 阶段 4

II.4.1 任务描述

第四阶段要求对递归调用的分析完成，由于涉及到栈的动态变化，这一阶段较为困难。

II.4.2 实验过程

- 在反汇编中跳转到 phase_4 的部分，反汇编代码如下 (已添加必要注释):

```
1  08048d22 <phase_4>:
2   8048d22:   83 ec 2c                sub    $0x2c,%esp
3
4  ///// parameter
5   8048d25:   8d 44 24 1c             lea    0x1c(%esp),%eax
6   8048d29:   89 44 24 0c             mov    %eax,0xc(%esp)
7
8   8048d2d:   8d 44 24 18             lea    0x18(%esp),%eax
9   8048d31:   89 44 24 08             mov    %eax,0x8(%esp)
10
11  8048d35:   c7 44 24 04 b1 a5 04    movl   $0x804a5b1,0x4(%esp)
12  8048d3c:   08
13
14  8048d3d:   8b 44 24 30             mov    0x30(%esp),%eax          // parameter
15  8048d41:   89 04 24                mov    %eax,(%esp)
16 /////
17  8048d44:   e8 17 fb ff ff          call   8048860 <__isoc99_sscanf@plt>
18
19  8048d49:   83 f8 02                cmp    $0x2,%eax
20  8048d4c:   75 07                   jne    8048d55 <phase_4+0x33>
21
22  8048d4e:   83 7c 24 18 0e          cmpl   $0xe,0x18(%esp)
23  8048d53:   76 05                   jbe    8048d5a <phase_4+0x38>   // a less than or
        equal to 0xe
24  8048d55:   e8 6b 05 00 00          call   80492c5 <explode_bomb>
25
26 ///// parameter
27  8048d5a:   c7 44 24 08 0e 00 00    movl   $0xe,0x8(%esp)
28  8048d61:   00
29  8048d62:   c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
30  8048d69:   00
31  8048d6a:   8b 44 24 18             mov    0x18(%esp),%eax
32  8048d6e:   89 04 24                mov    %eax,(%esp)
33 /////
34  8048d71:   e8 4b ff ff ff          call   8048cc1 <func4>
35
36  8048d76:   83 f8 06                cmp    $0x6,%eax
37  8048d79:   75 07                   jne    8048d82 <phase_4+0x60>
38
39  8048d7b:   83 7c 24 1c 06          cmpl   $0x6,0x1c(%esp)
40  8048d80:   74 05                   je     8048d87 <phase_4+0x65>
41
42  8048d82:   e8 3e 05 00 00          call   80492c5 <explode_bomb>
43
44  8048d87:   83 c4 2c                add    $0x2c,%esp
45  8048d8a:   c3                      ret
```

- 可以看出，程序执行了一次 scanf，并且第二个实参与 phase_3 中的实参一样，古这一阶段也要求输入两个整数。
- 在读入数字后，首先将第一个熟悉与 $0xe 进行比较 (0x8048d4e 处)，并且要求第一个参数小于或等于 0xe，否则炸弹爆炸。
- 然后调用了 func4 子程序，传入的参数依次为 scanf 获得的第一个数字、0、0xe。

- 在调用完毕后，首先将返回值与 $0x6 比较 (0x8048d76 处)，要求返回值等于 6，然后将 scanf 获得的第二个参数与 0x6 相比较 (0x8048d7b 处)，也要求相等。而在调用 func4 时未传入 scanf 获得的第二个参数，因此推测 scanf 获得的第二个参数不变，为 6。
- 对于 func4 进行分析，得到 func4 的流程图如图2.4.1所示。

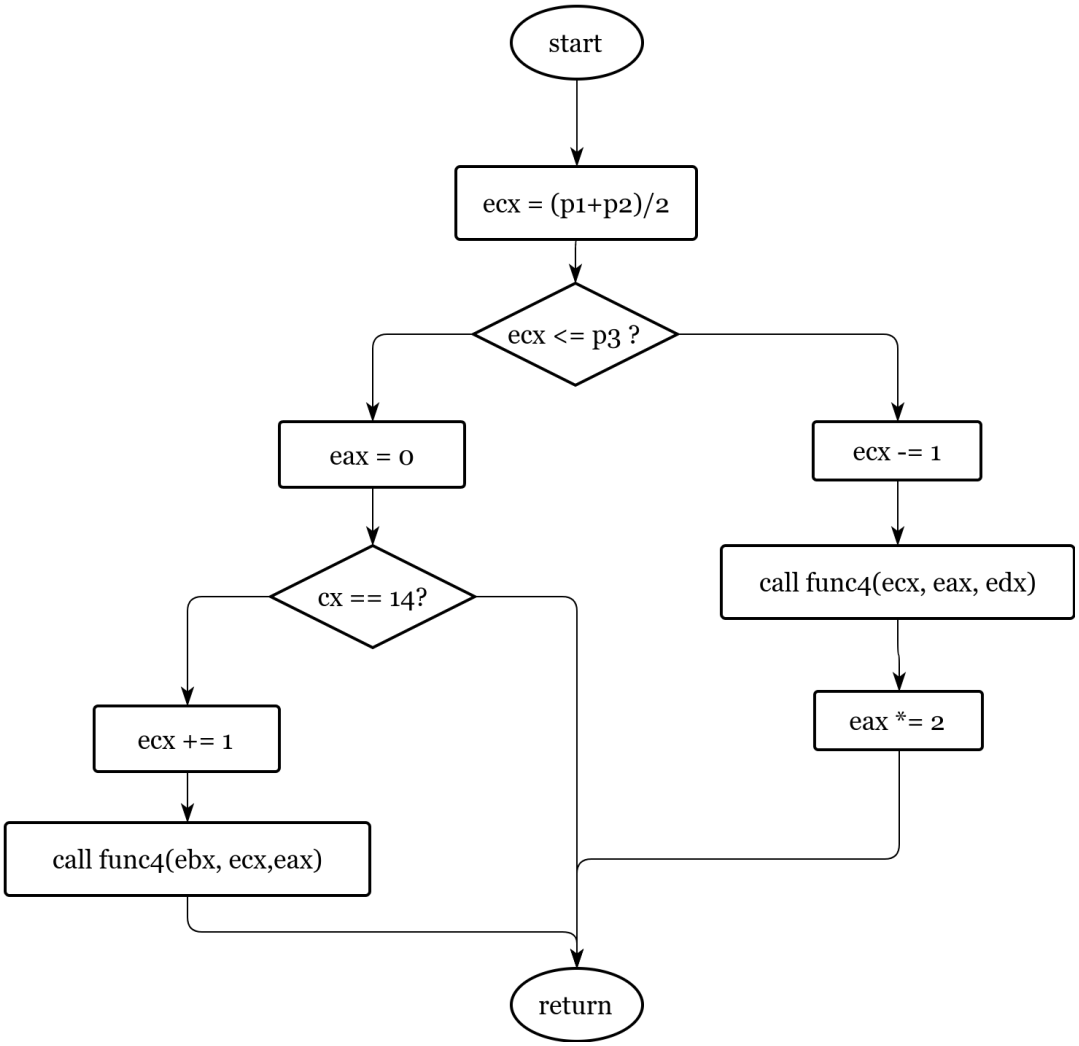

图 2.4.1: func4 流程图

- 由于 func4 的流程过于复杂，且带有递归，因此分析起来较为困难，故放弃分析。但是由 phase4 的已知条件可以知道第一个参数小于 14，第二个参数为 6，因此直接使用暴力手段进行拆解：从 0 开始对第一个参数逐个进行尝试，运行到 0x804d76 处观察 func4 的返回值是否为 6，若不为 6，则强制结束程序并使用下一个整数进行测试，直到测试的返回值为 6 为止。
- 通过上述测试，发现参数 1 的值为 6。图2.4.2展示了第四阶段拆除后的结果。

```
~/t/c/h/bomb807 cat result.txt                Wed 07 Jun 2017 03:13:51 PM CST
I am not part of the problem. I am a Republican.
1 2 4 8 16 32
1 -473
6 6
~/t/c/h/bomb807 ./bomb result.txt             Wed 07 Jun 2017 03:13:54 PM CST
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
```

图 2.4.2: 第四阶段拆除

II.4.3 实验结果

II.5 阶段 5

II.5.1 任务描述

第五阶段最要考察对于指针的分析。通过分析汇编代码语句对应的 C 语言指针操作来对炸弹进行拆除。

II.5.2 实验过程

- 首先跳转到 phase_5 的代码处，代码如下：

```
08048d8b <phase_5>:
 8048d8b:   83 ec 2c                sub    $0x2c,%esp

///// parameter
 8048d8e:   8d 44 24 1c             lea    0x1c(%esp),%eax
 8048d92:   89 44 24 0c             mov    %eax,0xc(%esp)

 8048d96:   8d 44 24 18             lea    0x18(%esp),%eax
 8048d9a:   89 44 24 08             mov    %eax,0x8(%esp)

 8048d9e:   c7 44 24 04 b1 a5 04    movl   $0x804a5b1,0x4(%esp)
 8048da5:   08

 8048da6:   8b 44 24 30             mov    0x30(%esp),%eax
 8048daa:   89 04 24                mov    %eax,(%esp)
/////
 8048dad:   e8 ae fa ff ff          call   8048860 <__isoc99_sscanf@plt>
 8048db2:   83 f8 01                cmp    $0x1,%eax
 8048db5:   7f 05                   jg     8048dbc <phase_5+0x31>
 8048db7:   e8 09 05 00 00          call   80492c5 <explode_bomb>


 8048dbc:   8b 44 24 18             mov    0x18(%esp),%eax
 8048dc0:   83 e0 0f                and    $0xf,%eax
 8048dc3:   89 44 24 18             mov    %eax,0x18(%esp)
 8048dc7:   83 f8 0f                cmp    $0xf,%eax
 8048dca:   74 2a                   je     8048df6 <phase_5+0x6b>   // BOOM!

```

```
29   8048dcc:   b9 00 00 00 00          mov    $0x0,%ecx
30   8048dd1:   ba 00 00 00 00          mov    $0x0,%edx
31
32   8048dd6:   83 c2 01                add    $0x1,%edx                // L1
33   8048dd9:   8b 04 85 a0 a3 04 08    mov    0x804a3a0(,%eax,4),%eax
34
35   8048de0:   01 c1                   add    %eax,%ecx
36   8048de2:   83 f8 0f                cmp    $0xf,%eax
37   8048de5:   75 ef                   jne    8048dd6 <phase_5+0x4b>   // jne L1 (loop)
38
39   8048de7:   89 44 24 18             mov    %eax,0x18(%esp)
40   8048deb:   83 fa 0f                cmp    $0xf,%edx
41   8048dee:   75 06                   jne    8048df6 <phase_5+0x6b>   // BOOM!
42
43   8048df0:   3b 4c 24 1c             cmp    0x1c(%esp),%ecx
44   8048df4:   74 05                   je     8048dfb <phase_5+0x70>
45
46   8048df6:   e8 ca 04 00 00          call   80492c5 <explode_bomb>
47
48   8048dfb:   83 c4 2c                add    $0x2c,%esp
```

- 对于汇编代码进行分析，发现首先调用了 sscanf 函数，且读入的格式化字符串地址与第三、第四阶段相同，均为 0x804a5b1，因此可以断定这一阶段的答案为两个整数。
- 对于接下来的代码进行分析，发现器为一个简单的循环。循环的出事条件为%eax = < 第一个参数 >&0xf(0x8048dc0 处)，结束条件为%eax 为 15(0x8048de2 处)。在这些循环中，每次去以 0x804a3a0 为数组首地址，以 eax 为索引的一个数 (0x8048dd9 处)，并存入 eax 中。并且，每次循环中 edx+=1(0x8048dd6 处)，ecx+=< 取出的数 >(0x8048de0 处)。
- 通过对跳出循环后的语句的分析，可以看到要求最后 edx 必须为 15(ox8048deb 处)，且 ecx 必须与 scanf 读入的第二个数相等。由此可以推断，第一个参数需要保证循环体恰好循环 15 次，且最后一次取出的值为 0xf，第二个参数则是取出的所有值的和。
- 对于循环中出现的数组进行反汇编，得到数组中的值依次为 0xa, 0x2, 0xe, 0x7, 0x8, 0xc, 0xf, 0xb, 0x0, 0x4, 0x1, 0xd, 0x3, 0x9, 0x6, 0x5，如图2.5.1所示。

```
~/t/c/h/bomb807 objdump --start-address 0x804a3a0 -s bomb | head

bomb:     file format elf32-i386

Contents of section .rodata:
 804a3a0 0a000000 02000000 0e000000 07000000  ................
 804a3b0 08000000 0c000000 0f000000 0b000000  ................
 804a3c0 00000000 04000000 01000000 0d000000  ................
 804a3d0 03000000 09000000 06000000 05000000  ................
 804a3e0 536f2079 6f752074 68696e6b 20796f75  So you think you
 804a3f0 2063616e 2073746f 70207468 6520626f   can stop the bo
```

图 2.5.1: 对于数组的反汇编

得到次数组后按照所得的规则进行逆推，得到第一个参数为 5，第二个参数为 115。输入炸弹，炸弹的第五阶段成功被拆除，如图2.5.2所示。

```
% ~/t/c/h/bomb807 cat result.txt                Wed 07 Jun 2017 03:47:57 PM CST
I am not part of the problem. I am a Republican.
1 2 4 8 16 32
1 -473
6 6
5 115
% ~/t/c/h/bomb807 ./bomb result.txt             Wed 07 Jun 2017 03:47:58 PM CST
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
```

图 2.5.2: 第五阶段拆除成功

II.5.3 实验结果

第五阶段的解为字符串 “5 115”

II.6 阶段 6

II.6.1 任务描述

实验六较为复杂, 要求通过对于汇编代码中的结构/链表操作进行分析并得出第六阶段的结果，对于结构体在内存中的存储方式进行了考察。

II.6.2 实验过程

- 首先跳转到 phase_6 的位置，代码如下 (由于 phase_6 的代码较长，因此分析时加入了较多的注释，===> 为在这一部分推断出的的限制条件):

```
1  08048d22 <phase_4>:
2   8048d22:   83 ec 2c              sub    $0x2c,%esp
3
4  ///// parameter
5   8048d25:   8d 44 24 1c           lea    0x1c(%esp),%eax
6   8048d29:   89 44 24 0c           mov    %eax,0xc(%esp)
7
8   8048d2d:   8d 44 24 18           lea    0x18(%esp),%eax
9   8048d31:   89 44 24 08           mov    %eax,0x8(%esp)
10
11  8048d35:   c7 44 24 04 b1 a5 04  movl   $0x804a5b1,0x4(%esp)
12  8048d3c:   08
13
14  8048d3d:   8b 44 24 30           mov    0x30(%esp),%eax          // parameter
15  8048d41:   89 04 24              mov    %eax,(%esp)
16 /////
17  8048d44:   e8 17 fb ff ff        call   8048860 <__isoc99_sscanf@plt>
18
19  8048d49:   83 f8 02              cmp    $0x2,%eax
20  8048d4c:   75 07                 jne    8048d55 <phase_4+0x33>
21
22  8048d4e:   83 7c 24 18 0e        cmpl   $0xe,0x18(%esp)
```

```
23  8048d53:   76 05                jbe    8048d5a <phase_4+0x38>   // a less than or
        equal to 0xe
24  8048d55:   e8 6b 05 00 00      call   80492c5 <explode_bomb>
25
26  ///// parameter
27  8048d5a:   c7 44 24 08 0e 00 00    movl   $0xe,0x8(%esp)
28  8048d61:   00
29  8048d62:   c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
30  8048d69:   00
31  8048d6a:   8b 44 24 18          mov    0x18(%esp),%eax
32  8048d6e:   89 04 24             mov    %eax,(%esp)
33  /////
34  8048d71:   e8 4b ff ff ff       call   8048cc1 <func4>
35
36  8048d76:   83 f8 06             cmp    $0x6,%eax
37  8048d79:   75 07                jne    8048d82 <phase_4+0x60>
38
39  8048d7b:   83 7c 24 1c 06       cmpl   $0x6,0x1c(%esp)
40  8048d80:   74 05                je     8048d87 <phase_4+0x65>
41
42  8048d82:   e8 3e 05 00 00       call   80492c5 <explode_bomb>
43
44  8048d87:   83 c4 2c             add    $0x2c,%esp
45  8048d8a:   c3                   ret
```

- 首先，粗略的对于代码进行分段，将代码分为五个部分 (注释中的 part1 part5) 并逐个进行分析。
- 第一部分：
  - 调用 read_six_numbers 读入六个数字。进行循环，将每一个数减一以后与 5 进行比较，要求 $\leqslant 5$，推断出读入的六个数中每一个都 $\leqslant 6$。进行第二个循环，这是一个二重循环，检测每一个数是否相等，如果不相等则炸弹爆炸，推测出读入的六个数两两不相等。
- 第二部分：
  - 进行一个循环，对于读入的每一个数进行处理，将 7-< 读入的数 > 存入一个数组的对应位置中，此列表在栈空间中并与读入的六个数的位置紧邻。
- 第三部分：
  - 这一部分较为复杂，也是较为重要的一部分。首先对其进行观察，发现它涉及到地址 0x804c154，从这个地址开始反汇编，结果如图2.6.1所示。

```
~/t/c/h/bomb807 objdump --start-address 0x804c154 -s bomb | head

bomb:     file format elf32-i386

Contents of section .data:
 804c154 c8030000 01000000 60c10408 b6030000  ........`.......
 804c164 02000000 6cc10408 8e010000 03000000  ....l...........
 804c174 78c10408 5b010000 04000000 84c10408  x...[...........
 804c184 ea000000 05000000 90c10408 86010000  ................
 804c194 06000000 00000000 00000000 55323031  ............U201
 804c1a4 35313438 39380000 00000000 00000000  514898..........
```

图 2.6.1: 对于 0x804c154 的反汇编

  - 显然，这是一个链表，链表中的每一个节点为一个结构，占 12 个字节，且第三个成员为 next 指针。
  - 经过分析后发现，这一部分进行的操作为对第二部分中的数组中的元素进行赋值。赋值规则为 tab[i+6] = chain + i，其中 chain 为链表的首地址。

- 第四部分：对于第三部分的排序后的链表进行了重新的链接。
- 第五部分：对于重新链接的链表进行了检查，要求保证链表从首至尾是升序有序的。整合上面的信息，输入的数据应该是对于原链表中的数据 0x3c8, 0x3b6, 0x18e, 0x15b, 0xea, 0x186 进行排序，并用 7 减去排序后的索引，然后进行倒序的结果。推算出应为 6 5 4 1 3 2，将 “6 5 4 1 3 2” 输入炸弹，成功拆除，如图2.6.2所示。

```
~/t/c/h/bomb807 ./bomb result.txt            Wed 07 Jun 2017 05:24:56 PM CST
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
~/t/c/h/bomb807                              Wed 07 Jun 2017 05:25:02 PM CST
```

图 2.6.2: 第六阶段成功拆除

II.6.3 实验结果

第六阶段的解为字符串 “6 5 4 1 3 2”

II.7 隐藏阶段

II.7.1 任务描述

找到如何进入隐藏阶段并对于隐藏阶段的炸弹进行拆解。

II.7.2 实验过程

- 在反汇编代码中全文搜索 phase 时，发现了 secrect_phase 函数，根据名称即可推测此为隐藏阶段，然后全文搜索其地址 8048f3e，发现其入口在 phase_defused 中。两个函数的反汇编代码如下。

  – phase_defused 函数：

```
0804944e <phase_defused>:
 804944e:   81 ec 8c 00 00 00       sub    $0x8c,%esp
 8049454:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
 804945a:   89 44 24 7c             mov    %eax,0x7c(%esp)
 804945e:   31 c0                   xor    %eax,%eax
 8049460:   c7 04 24 01 00 00 00    movl   $0x1,(%esp)
 8049467:   e8 4b fd ff ff          call   80491b7 <send_msg>
 804946c:   83 3d e8 c7 04 08 06    cmpl   $0x6,0x804c7e8
 8049473:   75 7e                   jne    80494f3 <phase_defused+0xa5>

///// parameter
 8049475:   8d 44 24 2c             lea    0x2c(%esp),%eax
 8049479:   89 44 24 10             mov    %eax,0x10(%esp)

 804947d:   8d 44 24 28             lea    0x28(%esp),%eax
 8049481:   89 44 24 0c             mov    %eax,0xc(%esp)

 8049485:   8d 44 24 24             lea    0x24(%esp),%eax
 8049489:   89 44 24 08             mov    %eax,0x8(%esp)

 804948d:   c7 44 24 04 0b a6 04    movl   $0x804a60b,0x4(%esp)

 8049494:   08
 8049495:   c7 04 24 f0 c8 04 08    movl   $0x804c8f0,(%esp)
/////
 804949c:   e8 bf f3 ff ff          call   8048860 <__isoc99_sscanf@plt>
 80494a1:   83 f8 03                cmp    $0x3,%eax
 80494a4:   75 35                   jne    80494db <phase_defused+0x8d>

///// parameter
 80494a6:   c7 44 24 04 14 a6 04    movl   $0x804a614,0x4(%esp)
 80494ad:   08
 80494ae:   8d 44 24 2c             lea    0x2c(%esp),%eax
 80494b2:   89 04 24                mov    %eax,(%esp)
/////
 80494b5:   e8 90 fb ff ff          call   804904a <strings_not_equal>
 80494ba:   85 c0                   test   %eax,%eax
 80494bc:   75 1d                   jne    80494db <phase_defused+0x8d>

 80494be:   c7 04 24 60 a4 04 08    movl   $0x804a460,(%esp)
 80494c5:   e8 26 f3 ff ff          call   80487f0 <puts@plt>
 80494ca:   c7 04 24 88 a4 04 08    movl   $0x804a488,(%esp)
 80494d1:   e8 1a f3 ff ff          call   80487f0 <puts@plt>
 80494d6:   e8 63 fa ff ff          call   8048f3e <secret_phase>

 80494db:   c7 04 24 c0 a4 04 08    movl   $0x804a4c0,(%esp)
 80494e2:   e8 09 f3 ff ff          call   80487f0 <puts@plt>
 80494e7:   c7 04 24 ec a4 04 08    movl   $0x804a4ec,(%esp)
 80494ee:   e8 fd f2 ff ff          call   80487f0 <puts@plt>
```

– secrect_phase 函数:

```
08048f3e <secret_phase>:
 8048f3e:   53                      push   %ebx
 8048f3f:   83 ec 18                sub    $0x18,%esp
 8048f42:   e8 0d 04 00 00          call   8049354 <read_line>

///// parameter
 8048f47:   c7 44 24 08 0a 00 00    movl   $0xa,0x8(%esp)
 8048f4e:   00
 8048f4f:   c7 44 24 04 00 00 00    movl   $0x0,0x4(%esp)
 8048f56:   00
 8048f57:   89 04 24                mov    %eax,(%esp)
/////
 8048f5a:   e8 71 f9 ff ff          call   80488d0 <strtol@plt>

 8048f5f:   89 c3                   mov    %eax,%ebx
 8048f61:   8d 40 ff                lea    -0x1(%eax),%eax              // eax -= 1
 8048f64:   3d e8 03 00 00          cmp    $0x3e8,%eax                  // eax ? 0
      x3e8
 8048f69:   76 05                   jbe    8048f70 <secret_phase+0x32>  // ===>
      para1 <= 0x3e9

 8048f6b:   e8 55 03 00 00          call   80492c5 <explode_bomb>

///// parameter
 8048f70:   89 5c 24 04             mov    %ebx,0x4(%esp)
 8048f74:   c7 04 24 a0 c0 04 08    movl   $0x804c0a0,(%esp)
/////
 8048f7b:   e8 6d ff ff ff          call   8048eed <fun7>

 8048f80:   85 c0                   test   %eax,%eax
 8048f82:   74 05                   je     8048f89 <secret_phase+0x4b>
 8048f84:   e8 3c 03 00 00          call   80492c5 <explode_bomb>

 8048f89:   c7 04 24 58 a3 04 08    movl   $0x804a358,(%esp)
 8048f90:   e8 5b f8 ff ff          call   80487f0 <puts@plt>
 8048f95:   e8 b4 04 00 00          call   804944e <phase_defused>
 8048f9a:   83 c4 18                add    $0x18,%esp
 8048f9d:   5b                      pop    %ebx
 8048f9e:   c3                      ret
 8048f9f:   90                      nop
```

- 首先对于 phase_defused 进行分析，发现其 strings_not_equal 函数的一个参数为地址 0x804a614。对于 bomb 进行反汇编，发现此处的字符串为 DrEvil，因此推测，进入隐藏阶段的字符串为 DrEvil。根据提示将起附加在 phase_4 的解后面，成功进入隐藏阶段，如图2.7.1所示。

```
% ~/t/c/h/bomb807 cat result.txt
I am not part of the problem. I am a Republican.
1 2 4 8 16 32
1 -473
6 6 DrEvil
5 115
6 5 4 1 3 2
% ~/t/c/h/bomb807 ./bomb result.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

图 2.7.1: 进入隐藏阶段

- 然后对于隐藏阶段的代码进行分析，发现首先读入一个字符串，然后将字符串按照 10 进制转换为长整形，并将这一长整形作为 fun7 的参数并调用 fun7。然后要求 fun7 的返回值为 0。其中 fun7 的反汇编代码如下：

```
08048eed <fun7>:
 8048eed:   53                      push   %ebx
 8048eee:   83 ec 18                sub    $0x18,%esp
 8048ef1:   8b 54 24 20             mov    0x20(%esp),%edx
 8048ef5:   8b 4c 24 24             mov    0x24(%esp),%ecx
 8048ef9:   85 d2                   test   %edx,%edx
 8048efb:   74 37                   je     8048f34 <fun7+0x47>      // BOOM

 8048efd:   8b 1a                   mov    (%edx),%ebx
 8048eff:   39 cb                   cmp    %ecx,%ebx
 8048f01:   7e 13                   jle    8048f16 <fun7+0x29>

 8048f03:   89 4c 24 04             mov    %ecx,0x4(%esp)
 8048f07:   8b 42 04                mov    0x4(%edx),%eax
 8048f0a:   89 04 24                mov    %eax,(%esp)
 8048f0d:   e8 db ff ff ff          call   8048eed <fun7>

 8048f12:   01 c0                   add    %eax,%eax
 8048f14:   eb 23                   jmp    8048f39 <fun7+0x4c>

 8048f16:   b8 00 00 00 00          mov    $0x0,%eax
 8048f1b:   39 cb                   cmp    %ecx,%ebx
 8048f1d:   74 1a                   je     8048f39 <fun7+0x4c>

 8048f1f:   89 4c 24 04             mov    %ecx,0x4(%esp)
 8048f23:   8b 42 08                mov    0x8(%edx),%eax
 8048f26:   89 04 24                mov    %eax,(%esp)
 8048f29:   e8 bf ff ff ff          call   8048eed <fun7>

```

```
30   8048f2e:   8d 44 00 01          lea    0x1(%eax,%eax,1),%eax
31   8048f32:   eb 05                jmp    8048f39 <fun7+0x4c>
32   8048f34:   b8 ff ff ff ff       mov    $0xffffffff,%eax
33
34   8048f39:   83 c4 18             add    $0x18,%esp
35   8048f3c:   5b                   pop    %ebx
36   8048f3d:   c3                   ret
```

- 对于 fun7 进行分析，发现其对存在于 0x804c0a0(作为参数传入 fun7) 的二叉树进行搜索，二叉树反汇编图2.7.2所示，在其中查找隐藏阶段的解的位置，然后进行回溯，若回溯的边链接的是左子树，则返回值乘以 2，若为右子树则乘以 2 再加上 1。

```
% ~/t/c/h/bomb807 objdump --start-address 0x804c0a0 -s bomb | head

bomb:     file format elf32-i386

Contents of section .data:
 804c0a0 24000000 acc00408 b8c00408 08000000  $...............
 804c0b0 dcc00408 c4c00408 32000000 d0c00408  ........2.......
 804c0c0 e8c00408 16000000 30c10408 18c10408  ........0.......
 804c0d0 2d000000 f4c00408 3cc10408 06000000  -.......<.......
 804c0e0 00c10408 24c10408 6b000000 0cc10408  ....$...k.......
 804c0f0 48c10408 28000000 00000000 00000000  H...(...........
% ~/t/c/h/bomb807
```

图 2.7.2: 二叉树的反汇编部分

- 对于反汇编的二叉树进行分析，可得二叉树的结构如图2.7.3所示。

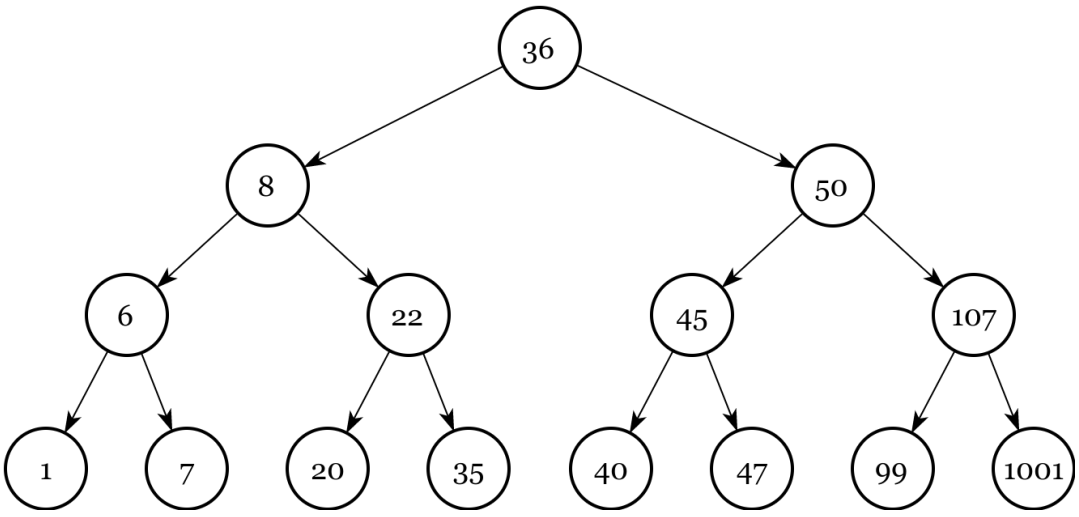

图 2.7.3: 二叉树结构

- 由图以及函数的调用规则可知，由于 secrect_phase 中 fun7 的返回值为 0，最后的答案为 1 或 6 或 8 或 36。
- 将 1 输入炸弹，成功拆除炸弹的隐藏部分，如图2.7.4所示。

```
~/t/c/h/bomb807 cat result.txt
I am not part of the problem. I am a Republican.
1 2 4 8 16 32
1 -473
6 6 DrEvil
5 115
6 5 4 1 3 2
1
~/t/c/h/bomb807 ./bomb result.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
~/t/c/h/bomb807
```

图 2.7.4: 炸弹的隐藏部分的拆除

II.7.3 实验结果

由实验过程可知，隐藏阶段的入口为在第四阶段的答案后追加 DrEvil，隐藏阶段的解为字符串 “1”。

III 实验小结

通过本次实验，我对于汇编，以及 C 语言如何转换为汇编有了更深入的了解。尤其第第 6 阶段，综合了各方面的指示，让我对于二进制程序的结构有了更深入的了解。相较于第一次实验，第二次实验增加了一定的难度，也使用了部分逆向工程的知识，让我更加深刻的体会到了这门课的魅力。