華中科技大學

课 程 实 验 报 告

课程名称： 计算机系统基础

实验名称： 二进制程序分析

院　　系：计算机科学与技术

专业班级：计算机 202201 班

学　　号：U202215357

姓　　名：王文涛

指导教师：朱　虹

2024 年 4 月 8 日

一、实验目的与要求

通过逆向分析一个二进制程序（称为“二进制炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示各方面知识点的理解，增强反汇编、跟踪、分析、调试等能力。

实验环境：Ubuntu，GCC，GDB 等

二、实验内容

作为实验目标的二进制炸弹（binary bombs）可执行程序由多个“关”组成。每一个“关”（阶段）要求输入一个特定字符串，如果输入满足程序代码的要求，该阶段即通过，否则程序输出失败。实验的目标是设法得到得出解除尽可能多阶段的字符串。

为了完成二进制炸弹的拆除任务，需要通过反汇编和分析跟踪程序每一阶段的机器代码，从中定位和理解程序的主要执行逻辑，包括关键指令、控制结构和相关数据变量等等，进而推断拆除炸弹所需要的目标字符串。

实验源程序及相关文件　bomb.rar

bomb.c　　主程序

phases.o　各个阶段的目标程序

support.c 完成辅助功能的目标程序

support.h　公共头文件

阶段 1：　串比较 phase_1(char *input);

要求输出的字符串(input) 与程序中内置的某一特定字符串相同。提示：找到与 input 串相比较的特定串的地址，查看相应单元中的内容，从而确定 input 应输入的串。

阶段 2：循环 phase_2(char *input);

要求在一行上输入 6 个整数数据，与程序自动产生的 6 个数据进行比较，若一致，则过关。提示：将输入串 input 拆分成 6 个数据由函数 read_six_numbers(input, numbers) 完成。之后是各个数据与自动产生的数据的比较，在比较中使用了循环语句。

阶段 3：条件分支 phase_3(char *input);

要求输入一个整数数据，该数据与程序自动生成的 一个数据比较，相等则过关。提示：在自动生成数据时，使用了 switch … case 语句。

阶段 4：递归调用和栈 phase_4(char *input);

要求在一行中输入两个数，第一个数表示在一个有序的数组（或者 binary search tree）中需要搜索到的数，该数是在一定范围之内的；第二个数表示找到搜索数的路径（在树的左边搜索编码为二进制位 0，在树的有边搜索编码为二进制位 1）。

阶段 5：指针和数组访问 phase_5(char *input);

要求在一行中输入一个串，该串与程序自动生成的串相同。在生成串和比较串时，使用了数组和指针。

阶段 6：链表、结构、指针的访问 phase_6(char *input);

要求在一行中输入 6 个数，这 6 个数是一个链表中结点的顺序号（从 1 到 6）。按照输入的顺序号，将对应链表结点中的值形成一个数组。若该数组是按照降序排列的，则过关。

三、实验记录及问题回答

1. 实验任务 1 的实验记录

首先，将程序反汇编，在 phase_1 前停下，要求输入字符串，先输入 aaaa。

```
Gate 1 :  input a string that meets the requirements.
aaaa
```

查看反汇编程序，发现程序会调用函数 string_not_equal。在调用函数前，压入了两个参数%eax，0x8(%ebp)。

```
push   %eax
push   0x8(%ebp)
call   0x80495c2 <strings_not_equal>
```

查看 eax 的内容, 是一个字符串的地址，字符串为 Instructions set architecture。

```
(gdb) i r eax
eax            0x804c1be          134529470
(gdb) x /s 0x804c1be
0x804c1be <special+350>:        "Instructions set architecture"
```

再查看 0x8(%ebp)，也是字符串的地址，内容为我的输入 aaaa。

```
(gdb) i r ebp
ebp            0xffffcff8         0xffffcff8
(gdb) x /w 0xffffcff8+0x8
0xffffd000:     0x0804c3d0
(gdb) x /s 0x804c3d0
0x804c3d0 <input_strings+80>:   "aaaa"
```

可以知道，程序将输入和字符串 Instructions set architecture 比较，一致则成功。输入 Instructions set architecture，过关。

```
Gate 1 :  input a string that meets the requirements.
Instructions set architecture
Phase 1 passed!
```

2. 实验任务 2 的实验记录

读入输入的六个数后，程序首先检查-0x24(%ebp)中的内容，即输入的第一个数不能小于 0

```
<phase_2+41>  mov    -0x24(%ebp),%eax
<phase_2+44>  test   %eax,%eax
<phase_2+46>  jns    0x804989e <phase_2+53>
<phase_2+48>  call   0x8049809 <explode_bomb>
```

接着程序将 0x804c375 处的一个字节移入 ebx

```
<phase_2+53>  mov    -0x24(%ebp),%eax
<phase_2+56>  movzbl 0x804c375,%edx
<phase_2+63>  movsbl %dl,%edx
<phase_2+66>  sub    $0x30,%edx
<phase_2+69>  cmp    %edx,%eax
```

通过查看内存，发现 0x804c375 处是 studentid+9，不难发现是我输入学号的最后一位的

ascii 码，减去 0x30 后即学号最后一位，所以第一个数是 7

```
0x804c36c <studentid>:  "U202215357"
(gdb) x /s 0x804c375
0x804c375 <studentid+9>:        "7"
```

同理，第二个数要求是学号倒数第二位 5

```
<phase_2+78>  mov    -0x20(%ebp),%eax
<phase_2+81>  movzbl 0x804c374,%edx
<phase_2+88>  movsbl %dl,%edx
<phase_2+91>  sub    $0x30,%edx
<phase_2+94>  cmp    %edx,%eax
```

接下来是一个循环，-0x28(%ebp)初始值为 2，每次加一，大于 5 时跳出循环，循环四次，每次将前两位相加与输入比较。所以后四位为 7+5=12,5+12=17,12+17=29,17+29=46

```
<phase_2+112> mov    -0x28(%ebp),%eax
<phase_2+115> mov    -0x24(%ebp,%eax,4),%eax
<phase_2+119> mov    -0x28(%ebp),%edx
<phase_2+122> sub    $0x1,%edx
<phase_2+125> mov    -0x24(%ebp,%edx,4),%ecx
<phase_2+129> mov    -0x28(%ebp),%edx
<phase_2+132> sub    $0x2,%edx
<phase_2+135> mov    -0x24(%ebp,%edx,4),%edx
<phase_2+139> add    %ecx,%edx
<phase_2+141> cmp    %edx,%eax
<phase_2+143> je     0x80498ff <phase_2+150>
<phase_2+145> call   0x8049809 <explode_bomb>
<phase_2+150> addl   $0x1,-0x28(%ebp)
<phase_2+154> cmpl   $0x5,-0x28(%ebp)
<phase_2+158> jle    0x80498d9 <phase_2+112>
```

输入 7 5 12 17 29 46，过关。

```
Gate 2 :  input six intergers that meets the requirements.
7 5 12 17 29 46
Phase 2 passed!
```

3. 实验任务 3 的实验记录

首先与第二关类似，要求第一个数是学号倒数第三位。

```
<phase_3+75>  movzbl 0x804c373,%eax
<phase_3+82>  movsbl %al,%eax
<phase_3+85>  lea    -0x30(%eax),%edx
<phase_3+88>  mov    -0x1c(%ebp),%eax
<phase_3+91>  cmp    %eax,%edx
<phase_3+93>  je     0x8049981 <phase_3+100>
<phase_3+95>  call   0x8049809 <explode_bomb>
```

接着是一个 switch 语句。当数大于 9 时，不符合要求，炸弹爆炸。将数字乘以 4 加 0x80499ec 存入 eax。查看内存 0x80499ec 发现储存的是连续的十个指令地址，对应不同的 case。

```
<phase_3+100> mov    -0x1c(%ebp),%eax
<phase_3+103> cmp    $0x9,%eax
<phase_3+106> ja     0x80499ec <phase_3+207>
<phase_3+108> mov    0x804a368(,%eax,4),%eax
<phase_3+115> jmp    *%eax
```

```
(gdb) x /10w 0x804a368
0x804a368:      0x08049992      0x0804999b      0x080499a4      0x080499ad
0x804a378:      0x080499b6      0x080499bf      0x080499c8      0x080499d1
0x804a388:      0x080499da      0x080499e3
```

因为我的数字是 3，对应的数字是 0x28e，即 654。

```
<phase_3+144> movl   $0x28e,-0x14(%ebp)
<phase_3+151> jmp    0x80499f1 <phase_3+212>
```

输入 3 654，过关。

```
Gate 3 :  input 2 intergers.
3 654
Phase 3 passed!
```

4. **实验任务 4 的实验记录**

在调用 func4 之前，压入了三个参数，14，0 和我们输入的第一个数，记为 func4(x,y,z)。func4(14,0,z)

```
<phase_4+89>  push   $0xe
<phase_4+91>  push   $0x0
<phase_4+93>  push   %eax
<phase_4+94>  call   0x8049a12 <func4>
```

分析 func4 函数。

设 0x10(%ebp)中为参数 x，0xc(%ebp)中为参数 y。shr $0x1f,%edx 取 x-y 符号位 sign(x-y)。最后 eax 中为[sign(x-y)+x-y]/2+y。记为 mid。

```
0x08049a18 <+6>:     mov    0x10(%ebp),%eax
0x08049a1b <+9>:     sub    0xc(%ebp),%eax
0x08049a1e <+12>:    mov    %eax,%edx
0x08049a20 <+14>:    shr    $0x1f,%edx
0x08049a23 <+17>:    add    %edx,%eax
0x08049a25 <+19>:    sar    %eax
0x08049a27 <+21>:    mov    %eax,%edx
0x08049a29 <+23>:    mov    0xc(%ebp),%eax
0x08049a2c <+26>:    add    %edx,%eax
```

将 mid 与输入 z 进行比较。

```
0x08049a34 <+34>:    cmp    0x8(%ebp),%eax
0x08049a37 <+37>:    jle    0x8049a55 <func4+67>
```

如果 z<mid，调用 func4(mid-1,y,z)

```
0x08049a39 <+39>:    mov    -0xc(%ebp),%eax
0x08049a3c <+42>:    sub    $0x1,%eax
0x08049a3f <+45>:    sub    $0x4,%esp
0x08049a42 <+48>:    push   %eax
0x08049a43 <+49>:    push   0xc(%ebp)
0x08049a46 <+52>:    push   0x8(%ebp)
0x08049a49 <+55>:    call   0x8049a12 <func4>
```

返回后，将 eax*2

```
0x08049a4e <+60>:    add    $0x10,%esp
0x08049a51 <+63>:    add    %eax,%eax
```

如果 z>mid，调用 func4(x,mid+1,z)

```
0x08049a5d <+75>:    mov    -0xc(%ebp),%eax
0x08049a60 <+78>:    add    $0x1,%eax
0x08049a63 <+81>:    sub    $0x4,%esp
0x08049a66 <+84>:    push   0x10(%ebp)
0x08049a69 <+87>:    push   %eax
0x08049a6a <+88>:    push   0x8(%ebp)
0x08049a6d <+91>:    call   0x8049a12 <func4>
```

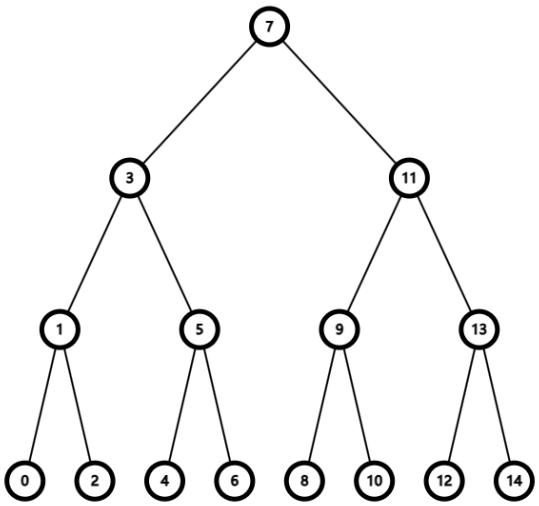
返回后将 eax*2+1

```
0x08049a72 <+96>:    add    $0x10,%esp
0x08049a75 <+99>:    add    %eax,%eax
0x08049a77 <+101>:   add    $0x1,%eax
```

将汇编代码翻译为 c 语言，为

```c
int func4(int x,int y,int z){
    int mid;
    if(x>y)
        mid=(x+y)/2;
    else
        mid=(x+y+1)/2;
    if(z<mid)
        return func4(mid-1,y,z)*2;
    else if(z>mid)
        return func4(x,mid+1,z)*2+1;
    else return 0;
}
```

不难看出，这是一个二分搜索，画出二叉树如下图所示。



由代码可知函数返回值只能为 7

```
<phase_4+113> mov    -0x1c(%ebp),%eax
<phase_4+116> cmp    %eax,-0x14(%ebp)
<phase_4+119> je     0x8049b01 <phase_4+126>
<phase_4+121> call   0x8049809 <explode_bomb>
```

由于二叉树只有三层，最大值为 7，所以要搜索的数一定是 14。

输入 14，7，过关。

```
Gate 3 :  input 2 intergers.
14 7
Phase 3 passed!
```

5. **实验任务 5 的实验记录**

首先判断输入是否为 6 个。

```
<phase_5+37>  mov    %eax,-0x18(%ebp)
<phase_5+40>  cmpl   $0x6,-0x18(%ebp)
<phase_5+44>  je     0x8049b48 <phase_5+51>
<phase_5+46>  call   0x8049809 <explode_bomb>
```

接着进入循环，从 0 到 5 循环。

edx 中储存循环次数，eax 中储存的是输入的字符串，取最后一个字节，依次从前往后储存。

```
<phase_5+60>  mov    -0x1c(%ebp),%edx
<phase_5+63>  mov    -0x2c(%ebp),%eax
<phase_5+66>  add    %edx,%eax
<phase_5+68>  movzbl (%eax),%eax
<phase_5+71>  movsbl %al,%eax
<phase_5+74>  and    $0xf,%eax
```

```
(gdb) i r eax
eax            0x804c510           134530320
(gdb) x /s 0x804c510
0x804c510 <input_strings+400>:  "mfcdhg"
```

0x804c350 处储存的是一个字符数组，eax 为下标，取对应的字符的最后一个字节存到一个空指针处。

```
<phase_5+77>  movzbl 0x804c350(%eax),%eax
<phase_5+84>  lea    -0x13(%ebp),%ecx
<phase_5+87>  mov    -0x1c(%ebp),%edx
<phase_5+90>  add    %ecx,%edx
<phase_5+92>  mov    %al,(%edx)
```

```
(gdb) x /s 0x804c350
0x804c350 <array.0>:    "maduiersnfotvbyl
```

循环结束后生成了一个字符串。接着进行字符串比较，压入了两个参数。

eax 中为生成的字符串，0x804a390 中为目标字符串。

```
<phase_5+111> push   $0x804a390
<phase_5+116> lea    -0x13(%ebp),%eax
<phase_5+119> push   %eax
<phase_5+120> call   0x80495c2 <strings not equal>
```

```
(gdb) x /s 0x804a390
0x804a390:      "bruins"
```

寻找每个字符在字符串 maduiersnfotvbyl 中的位置，得到答案为 d 6 3 4 8 7。

输入对应字符串 mfcdhg，通过。

```
Gate 5 :  input  a string.
mfcdhg
Phase 5 passed!
```

6. **实验任务 6 的实验记录**

读入六个数之后进入一个循环，判断六个数是否是数子 1 到 6 的一个排列。

接着进入另一个循环。

查看 eax 中储存地址的内容，发现是第一个节点 node1，每个结构体占 12 个字节，第一个是数，第二个是序号，第三个是指向下一个节点的指针，依次查看节点。

```
(gdb) x /3xw 0x804c290
0x804c290 <node1>:      0x00000119      0x00000001      0x0804c284
(gdb) x /3xw 0x804c284
0x804c284 <node2>:      0x0000038b      0x00000002      0x0804c278
(gdb) x /3xw 0x804c278
0x804c278 <node3>:      0x00000142      0x00000003      0x0804c26c
(gdb) x /3xw 0x804c26c
0x804c26c <node4>:      0x00000079      0x00000004      0x0804c260
(gdb) x /3xw 0x804c260
0x804c260 <node5>:      0x00000338      0x00000005      0x0804c254
(gdb) x /3xw 0x804c254
0x804c254 <node6>:      0x00000210      0x00000006      0x00000000
```

按数的大小从大到小排序为

输入 2 5 6 3 1 4，过关。

```
Gate 6 :  input 6 intergers.
2 5 6 3 1 4
Phase 6 passed!
```

四、体会

经过这次实验，我对汇编的掌握程度得到了巨大的提升。刚开始第一关的时候，看完 ppt 再做还是有点不明不白，但是经过对反汇编程序的研究，我已经能完全看懂反汇编程序的逻辑了，通过一步步地运行，查看寄存器和内存的值，以及画出栈帧图，能够理解程序的运行过程，进而过关。经过实验，我对 gdb 的使用也更加熟练了，调试的速度得到了极大的提升。