

哈尔滨工业大学

实验报告

实验（三）

题 目 二进制炸弹 Binary Bomb

专 业 人工智能

学 号 2021112845

班 级 2103601

学 生 张智雄

指 导 老 师 郑贵滨

实 验 地 点 G.709

实 验 日 期 2023.4.2

计算学部

目 录

第 1 章 实验基本信息	- 3 -
1.1 实验目的.....	- 3 -
1.2 实验环境与工具.....	- 3 -
1.2.1 硬件环境.....	- 3 -
1.2.2 软件环境.....	- 3 -
1.2.3 开发工具.....	- 3 -
1.3 实验预习.....	- 3 -
第 2 章 实验环境建立	- 4 -
2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分）	- 4 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分）	- 4 -
第 3 章 各阶段炸弹破解与分析	- 5 -
3.1 阶段 1 的破解与分析.....	- 5 -
3.2 阶段 2 的破解与分析.....	- 5 -
3.3 阶段 3 的破解与分析.....	- 7 -
3.4 阶段 4 的破解与分析.....	- 7 -
3.5 阶段 5 的破解与分析.....	- 9 -
3.6 阶段 6 的破解与分析.....	- 10 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 10 -
第 4 章 总结	- 11 -
4.1 请总结本次实验的收获.....	- 11 -
4.2 请给出对本次实验内容的建议.....	- 11 -
参考文献.....	- 11 -

第 1 章 实验基本信息

1.1 实验目的

熟练掌握计算机系统的 ISA 指令系统与寻址方式
熟练掌握 Linux 下调试器的反汇编调试跟踪分析机器语言的方法
增强对程序机器级表示、汇编语言、调试器和逆向工程等的理解

1.2 实验环境与工具

1.2.1 硬件环境

X64 CPU; 2.30GHz; 16G RAM; 1.5THD disk

1.2.2 软件环境

Windows11 64 位; VMware Workstation 17 Pro; Ubuntu 22.10

1.2.3 开发工具

Visual Studio 2019 64 位; CodeBlocks 64 位; vim+gcc

1.3 实验预习

1. 上实验课前, 必须认真预习实验指导书 (PPT 或 PDF), 了解实验的目的、实验环境与软硬件工具、实验操作步骤, 复习与实验有关的理论知识。
2. 写出 C 语言下包含字符串比较、循环、分支 (含 switch)、函数调用、递归、指针、结构、链表等的例子程序 sample.c。
3. 生成执行程序 sample.out。
4. 用 gcc -S 或 CodeBlocks 或 GDB 或 OBJDUMP 等, 反汇编, 比较。
5. 列出每一部分的 C 语言对应的汇编语言。
6. 修改编译选项-O (缺省 2)、O0、O1、O3、Og、-m32/m64。再次查看生成的汇编语言与原来的区别。
7. 注意 O1 之后缺省无栈帧, RBP 为普通寄存器。用 -fno-omit-frame-pointer 加上栈指针。
8. GDB 命令详解 - tui 模式 ^XA 切换 layout 改变等等
9. 有目的地学习: 看 VS 的功能, GDB 命令用什么?

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 反汇编 (10 分)

CodeBlocks 运行 hello.c。反汇编查看 printf 函数的实现。

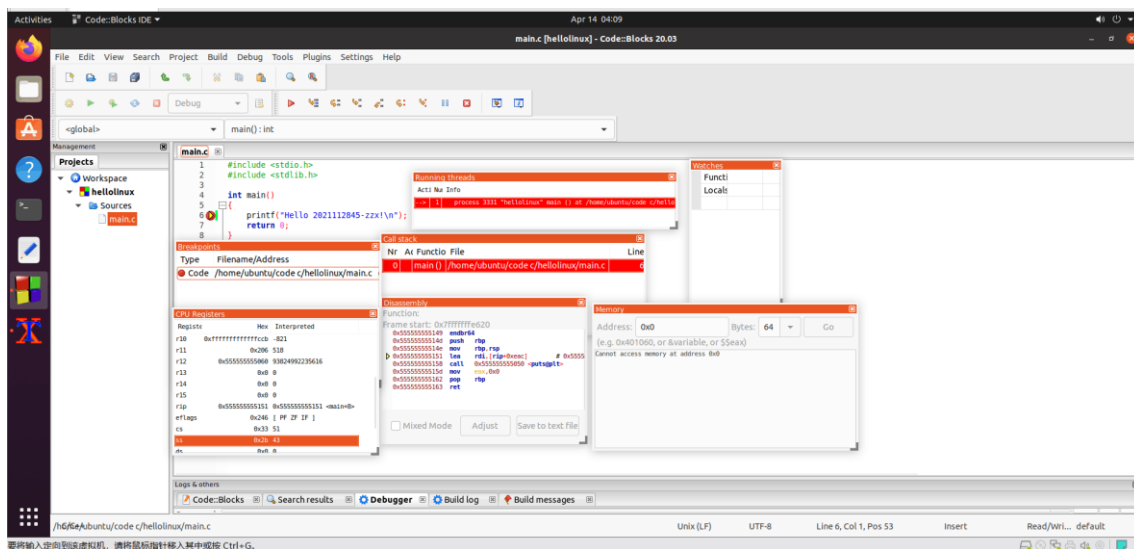


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

2.2 Ubuntu 下 EDB 运行环境建立 (10 分)

用 EDB 调试 hello.c 的执行文件，截图，要求同 2.1

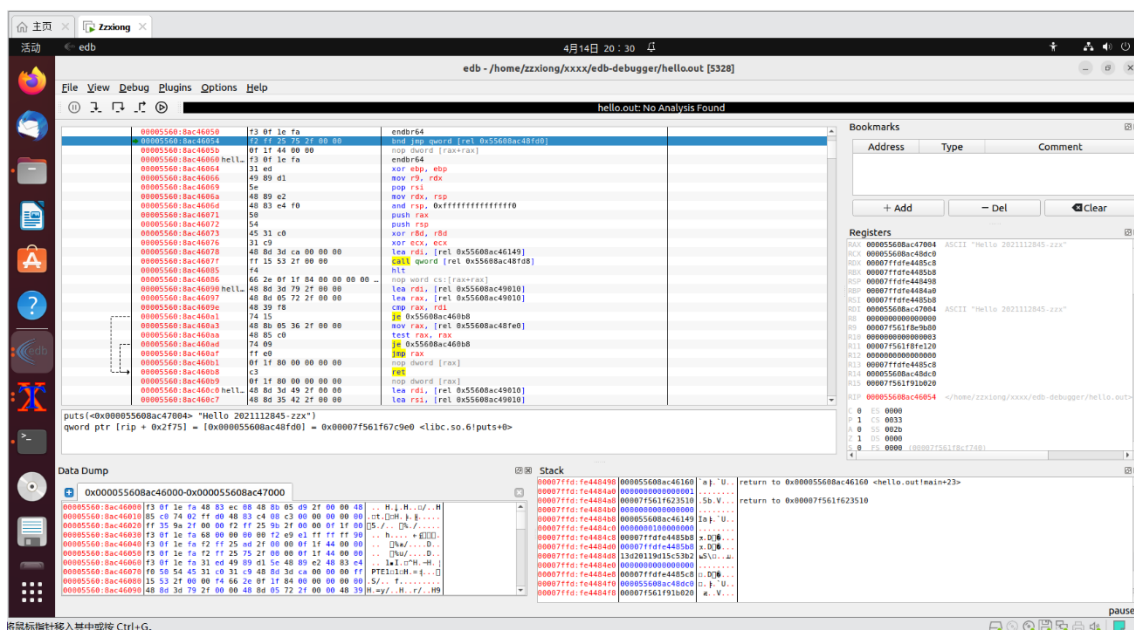


图 2-2 Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 30 分，密码 10 分，分析 20 分，总分不超过 80 分

3.1 阶段 1 的破解与分析

密码如下：All your base are belong to us.

破解过程：首先反汇编阶段 1 的函数代码，如下图 3-1-1 所示，观察分析得到此函数的运行逻辑：输入一个字符串，若与预设字符串不相同，则炸弹爆炸。

```
0x00000000004013f9 <+0>:    push    %rbp
0x00000000004013fa <+1>:    mov     %rsp,%rbp
0x00000000004013fd <+4>:    mov     $0x403150,%esi
0x0000000000401402 <+9>:    call    0x4017f6 <strings_not_equal>
0x0000000000401407 <+14>:   test    %eax,%eax
0x0000000000401409 <+16>:   jne     0x40140d <phase_1+20>
0x000000000040140b <+18>:   pop     %rbp
0x000000000040140c <+19>:   ret
0x000000000040140d <+20>:   call    0x4018f2 <explode_bomb>
0x0000000000401412 <+25>:   jmp     0x40140b <phase_1+18>
End of assembler dump.
```

图 3-1-1 phase_1 反汇编代码

注意到在调用函数 strings_not_equal 时，地址 0x403150 传入作为函数参数，推测答案存放在此地址中，直接查看该地址的字符串，得到答案。

```
(gdb) print (char*)0x403150
$1 = 0x403150 "All your base are belong to us."
```

图 3-1-2 phase_1 答案

3.2 阶段 2 的破解与分析

密码如下：1 2 4 8 16 32 64

破解过程：首先反汇编阶段 2 的函数代码，如下图 3-2-1 所示，注意到在 0x401421 地址处调用函数 read_six_numbers，推测本关输入为 6 个数（起始地址为 -x30(%rbp)）。

随后比较第一个数 -x30(%rbp) 与 1，若不等于 1，则炸弹引爆，于是第一个数解得为 1。而后初始化 %ebx 为 1，并作为循环标志量 (int i = 1; i <= 5; i++)。

```

Dump of assembler code for function phase_2:
0x0000000000401414 <+0>:    push    %rbp
0x0000000000401415 <+1>:    mov     %rsp,%rbp
0x0000000000401418 <+4>:    push    %rbx
0x0000000000401419 <+5>:    sub     $0x28,%rsp
0x000000000040141d <+9>:    lea     -0x30(%rbp),%rsi
0x0000000000401421 <+13>:   call    0x401914 <read_six_numbers>
0x0000000000401426 <+18>:   cmpl    $0x1,-0x30(%rbp)
0x000000000040142a <+22>:   jne     0x401433 <phase_2+31>
0x000000000040142c <+24>:   mov     $0x1,%ebx
0x0000000000401431 <+29>:   jmp     0x401442 <phase_2+46>
0x0000000000401433 <+31>:   call    0x4018f2 <explode_bomb>
0x0000000000401438 <+36>:   jmp     0x40142c <phase_2+24>
0x000000000040143a <+38>:   call    0x4018f2 <explode_bomb>
0x000000000040143f <+43>:   add     $0x1,%ebx
0x0000000000401442 <+46>:   cmp     $0x5,%ebx
0x0000000000401445 <+49>:   jg      0x40145d <phase_2+73>
0x0000000000401447 <+51>:   movslq  %ebx,%rdx
0x000000000040144a <+54>:   lea     -0x1(%rbx),%eax
0x000000000040144d <+57>:   cltq
0x000000000040144f <+59>:   mov     -0x30(%rbp,%rax,4),%eax
0x0000000000401453 <+63>:   add     %eax,%eax
0x0000000000401455 <+65>:   cmp     %eax,-0x30(%rbp,%rdx,4)

```

图 3-2-1 phase_2 反汇编代码

分析循环内部反汇编代码知，该程序检查首项为 1，项数为 6，公比为 2 的等比数列，可等价于如下代码：

```

int arg[6]; //输入的6个参数
if(arg[0] != 1)
    explode_bomb();
for(int i = 1; i <= 5; i++)
{
    int eax = arg[i-1];
    int edx = arg[i]; //-0x30(%rbp,%rdx,4)
    if(eax + eax != edx)
        explode_bomb();
}

```

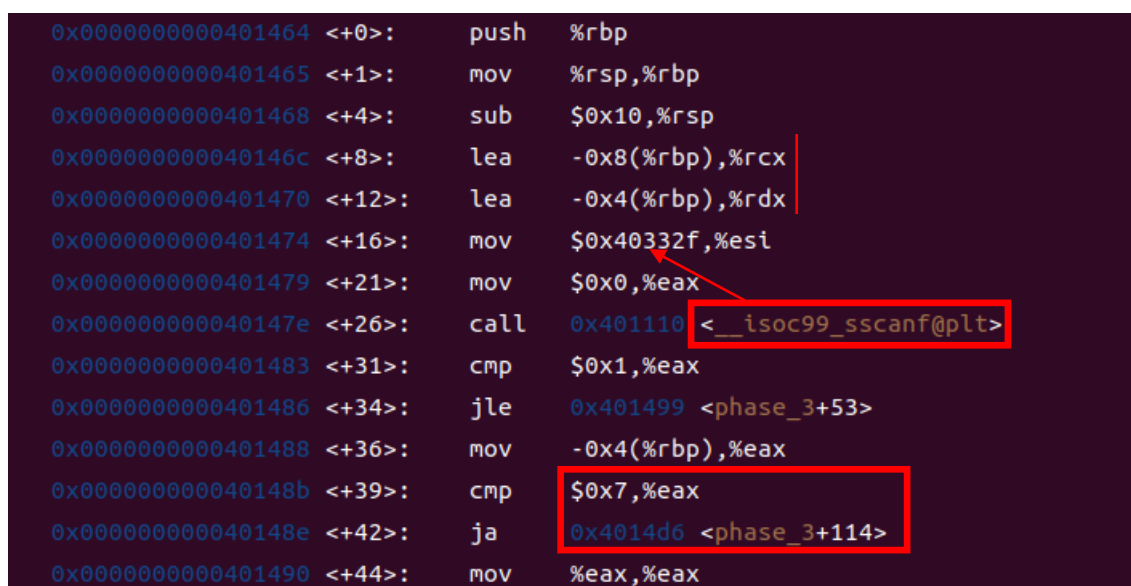
图 3-2-2 phase_2 等价代码

故答案为 1 2 4 8 16 32。

3.3 阶段 3 的破解与分析

密码如下：0 88 / 1 298 / 2 788 / 3 66 / 4 535 / 5 273 / 6 77 / 7 429

破解过程：首先反汇编阶段 3 的函数代码，截取部分如下图 3-3-1 所示，首先观察到输入的参数有两个，而后检查输入要求，得到输入要求为“%d %d”，分别储存在-0x8(%rbp)和-0x4(%rbp)中，同时注意到第一个参数-0x4(%rbp)必须 ≤ 7 ，否则炸弹引爆。



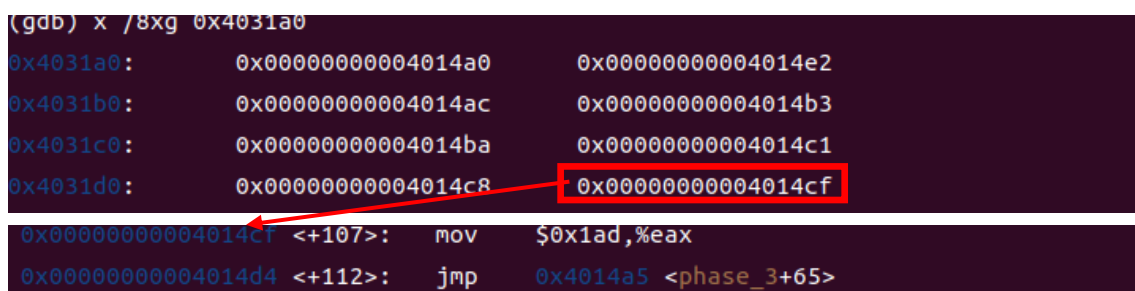
```

0x0000000000401464 <+0>:    push    %rbp
0x0000000000401465 <+1>:    mov     %rsp,%rbp
0x0000000000401468 <+4>:    sub     $0x10,%rsp
0x000000000040146c <+8>:    lea     -0x8(%rbp),%rcx
0x0000000000401470 <+12>:   lea     -0x4(%rbp),%rdx
0x0000000000401474 <+16>:   mov     $0x40332f,%esi
0x0000000000401479 <+21>:   mov     $0x0,%eax
0x000000000040147e <+26>:   call    0x401110 <__isoc99_sscanf@plt>
0x0000000000401483 <+31>:   cmp     $0x1,%eax
0x0000000000401486 <+34>:   jle     0x401499 <phase_3+53>
0x0000000000401488 <+36>:   mov     -0x4(%rbp),%eax
0x000000000040148b <+39>:   cmp     $0x7,%eax
0x000000000040148e <+42>:   ja      0x4014d6 <phase_3+114>
0x0000000000401490 <+44>:   mov     %eax,%eax

```

图 3-3-1 phase_3 反汇编代码

而后查看跳转表，跳转至相应地址可以发现会返回一个整数至%eax，即为第二个参数。故答案为 0 88 / 1 298 / 2 788 / 3 66 / 4 535 / 5 273 / 6 77 / 7 429



```

(gdb) x /8xg 0x4031a0
0x4031a0:    0x00000000004014a0    0x00000000004014e2
0x4031b0:    0x00000000004014ac    0x00000000004014b3
0x4031c0:    0x00000000004014ba    0x00000000004014c1
0x4031d0:    0x00000000004014c8    0x00000000004014cf
0x00000000004014cf <+107>:  mov     $0x1ad,%eax
0x00000000004014d4 <+112>:  jmp     0x4014a5 <phase_3+65>

```

图 3-3-1 phase_3 跳转表示意

3.4 阶段 4 的破解与分析

密码如下：108 2 / 162 3 / 216 4

破解过程：首先反汇编阶段 4 的函数代码，截取部分如下图 3-4-1 所示，观察

到输入的参数有两个，分别储存在 $-0x8(\%rbp)$ 和 $-0x4(\%rbp)$ 中，注意到第二个输入值 x （地址为 $-0x4(\%rbp)$ ）的取值范围为 $(1,4]$ ，即可取 $2,3,4$ 三个值，而后将 x 与第二个参数 $y = 8$ 传入 `func4` 函数，比较最后的返回值`%eax` 与第一个输入值（地址为 $-0x8(\%rbp)$ ）进行比较，若不相同则炸弹爆炸。

```

0x000000000040155f <+36>:  mov    -0x4(%rbp),%eax
0x0000000000401562 <+39>:  cmp     $0x1,%eax
0x0000000000401565 <+42>:  jle     0x40156c <phase_4+49>
0x0000000000401567 <+44>:  cmp     $0x4,%eax
0x000000000040156a <+47>:  jle     0x401571 <phase_4+54>
0x000000000040156c <+49>:  call    0x4018f2 <explode_bomb>
0x0000000000401571 <+54>:  mov     -0x4(%rbp),%esi
0x0000000000401574 <+57>:  mov     $0x8,%edi
0x0000000000401579 <+62>:  call    0x4014f0 <func4>
0x000000000040157e <+67>:  cmp     %eax,-0x8(%rbp)
0x0000000000401581 <+70>:  jne     0x401585 <phase_4+74>

```

图 3-4-1 phase_4 反汇编代码

而后查看`func4(x,y)`的反汇编代码如下图 3-4-2 所示，可以看到当 $y = 1$ 时函数返回值为 x ；当 $y = 0$ 时，函数返回值为 0 。

```

0x00000000004014f0 <+0>:  test    %edi,%edi
0x00000000004014f2 <+2>:  jle     0x401531 <func4+65>
0x0000000000401501 <+17>:  mov     %edi,%r12d
0x0000000000401504 <+20>:  mov     %esi,%ebx
0x0000000000401506 <+22>:  cmp     $0x1,%edi
0x0000000000401509 <+25>:  je      0x401537 <func4+71>
0x000000000040150b <+27>:  lea     -0x1(%rdi),%edi
0x000000000040150e <+30>:  call    0x4014f0 <func4>
0x0000000000401513 <+35>:  lea     (%rax,%rbx,1),%r13d
0x0000000000401517 <+39>:  lea     -0x2(%r12),%edi
0x000000000040151c <+44>:  mov     %ebx,%esi
0x000000000040151e <+46>:  call    0x4014f0 <func4>
0x0000000000401523 <+51>:  add     %r13d,%eax
0x0000000000401531 <+65>:  mov     $0x0,%eax
0x0000000000401537 <+71>:  mov     %esi,%eax

```

递归主体

图 3-4-2 func4 反汇编代码

分析递归主体得到 $func4(x,y) = func4(x,y-1) + func4(x,y-2) + x$ 。

运行函数程序得 $func4(2,8) = 108$, $func4(3,8) = 162$, $func4(4,8) = 216$ 。
故答案为 108 2 / 162 3 / 216 4。

3.5 阶段 5 的破解与分析

密码如下：llllla（答案不唯一）

破解过程：首先反汇编阶段 5 的函数代码，截取部分如下图 3-5-1 所示，观察到函数开始时调用了 `string_length` 函数，且若返回值不为 6，则炸弹爆炸，因而输入为一个长度为 6 的字符串。同时注意到在函数包含一个从 0 到 5 的一个循环节，而在循环节中，将 `%rbx` 数组内的数据与 0xf 做按位与运算，将结果保存在 `%edx` 中，同时提取 0x4031e0 地址下数组内偏移量为 `%edx` 的数据进行累加，若最后累加结果不为 0x41（十进制 65），则炸弹爆炸。

```

0x0000000000401595 <+9>:    mov    %rdi,%rbx
0x0000000000401598 <+12>:    call   0x4017e2 <string_length>
0x000000000040159d <+17>:    cmp    $0x6,%eax
0x00000000004015a0 <+20>:    jne     0x4015c7 <phase_5+59>
0x00000000004015a2 <+22>:    mov    $0x0,%ecx
0x00000000004015a7 <+27>:    mov    $0x0,%eax
0x00000000004015ac <+32>:    cmp    $0x5,%eax
0x00000000004015af <+35>:    jg      0x4015ce <phase_5+66>
0x00000000004015b1 <+37>:    movslq %eax,%rdx
0x00000000004015b4 <+40>:    movzbl (%rbx,%rdx,1),%edx
0x00000000004015b8 <+44>:    and     $0xf,%edx
0x00000000004015bb <+47>:    add     0x4031e0(,%rdx,4),%ecx
0x00000000004015c2 <+54>:    add     $0x1,%eax
0x00000000004015c5 <+57>:    jmp     0x4015ac <phase_5+32>
0x00000000004015c7 <+59>:    call   0x4018f2 <explode_bomb>
0x00000000004015cc <+64>:    jmp     0x4015a2 <phase_5+22>
0x00000000004015ce <+66>:    cmp     $0x41,%ecx
0x00000000004015d1 <+69>:    jne     0x4015da <phase_5+78>
  
```

图 3-4-1 phase_4 反汇编代码

0x4031e0 <array.3401>:	2	10	6	1
0x4031f0 <array.3401+16>:	12	16	9	3
0x403200 <array.3401+32>:	4	7	14	5
0x403210 <array.3401+48>:	11	8	15	13

图 3-4-2 0x4031e0 地址下数组数据

因此，`phase_5` 函数可等价于下图代码，只需输入满足程序要求的 6 位字符串即可拆除炸弹，llllla 为其中之一（答案不唯一）。

```
char string[6]; //长度为6的字符串
int data[16]; //0x4031e0数组
int sum = 0;
for(int i = 0; i <= 5; i++)
{
    int index = string[i] & 0xf;
    sum += data[index];
}
if(sum != 0x41) explode_bomb();
```

图 3-5-3 phase_5 等价代码

3.6 阶段 6 的破解与分析

密码如下：

破解过程：

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：

破解过程：

第 4 章 总结

4.1 请总结本次实验的收获

1. 了解了计算机系统的 ISA 指令系统与寻址方式，加深了对计算机指令的理解。
2. 熟悉了在 Linux 系统下使用 GDB 对程序进行调试分析，及利用反汇编调试跟踪分析机器语言的方法，加深了对程序运行中计算机各部分如内存、堆栈的工作原理的理解。
3. 熟悉了汇编语言，简单了解了高级语言到汇编语言编译过程，并结合具体实验巩固了程序的机器级表示相关知识。

4.2 请给出对本次实验内容的建议

1. 对 EDB 的安装及使用的教程和讲述不是很明确
2. 增加 PPT 部分内容的注解，有时不太能理解 PPT 的指令的用途及功能

注：本章为酌情加分项。

参考文献

[1] RANDALE.BRYANT, DAVIDR.O'HALLARON. 深入理解计算机系统[M]. 机械工业出版社, 2011.