北京郵電大學

实验报告

题　　目：拆解二进制炸弹

班　　级：2021211301

学　　号：2021210967

姓　　名：王心雨

学　　院：计算机学院

2022 年 10 月 24 日

一、实验目的

1. 理解 C 语言程序的机器级表示。
2. 初步掌握 GDB 调试器的用法。
3. 阅读 C 编译器生成的 x86-64 机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

二、实验环境

1. Windows PowerShell（10.120.11.12）；
2. Linux 系统；
3. Objdump 命令反汇编；
4. GDB 调试工具；
5. Vi 编辑器。

三、实验内容

登录 bupt1 服务器，在 home 目录下可以找到 Evil 博士专门为你量身定制的一个 bomb，当运行时，它会要求你输入一个字符串，如果正确，则进入下一关，继续要求你输入下一个字符串；否则，炸弹就会爆炸，输出一行提示信息并向计分服务器提交扣分信息。因此，本实验要求你必须通过反汇编和逆向工程对 bomb 执行文件进行分析，找到正确的字符串来解除这个的炸弹。

本实验通过要求使用课程所学知识拆除一个“binary bombs”来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。 “binary bombs”是一个 Linux 可执行程序，包含了 5 个阶段（或关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信；否则炸弹“爆炸”，打印输出 “BOOM!!!”。炸弹的每个阶段考察了机器级程序语言的一个不同方面，难度逐级递增。

为完成二进制炸弹拆除任务，需要使用 gdb 调试器和 objdump 来反汇编 bomb 文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验 2 的具体内容见实验 2 说明。

三、 实验步骤及实验分析

● 实验一

1. 打开 Windows PowerShell,输入 ssh 账号密码登录 linux 系统，输入“ll”指令打开当前文件目录。根据时间显示，看到被分配到了 bomb33，文件目录中多了一个 bomb33.tar 的压缩包文件（图 1）；再将压缩包文件解压，看到其中包括 bomb.c、bomb、README 三个文件（图 2）；

```
2021210967@bupt1:~$ ll
total 88
-rw-r--r-- 1 root       root     40960 Oct 18 17:36 bomb33.tar
-rwxr-xr-x 1 2021210967 students 18952 Oct 15 15:29 main*
-rw-r--r-- 1 2021210967 students   183 Oct 15 15:28 main.c
-rw-r--r-- 1 2021210967 students 10210 Oct 15 15:35 main.d
-rw-r--r-- 1 2021210967 students 10954 Oct 10 23:41 :wq
```

```
2021210967@bupt1:~$ tar -xvf bomb33.tar
bomb33/README
bomb33/bomb.c
bomb33/bomb
2021210967@bupt1:~$ ls
bomb33  bomb33.tar  main  main.c  main.d  :wq
2021210967@bupt1:~$ vi bomb.c
2021210967@bupt1:~$ 2021210967@bupt1:~$ vi main.c
2021210967@bupt1:~$ cd bomb33
2021210967@bupt1:~/bomb33$ ls
bomb  bomb.c  README
```

2. 利用 vi 编辑器打开并查看 bomb.c 文件源代码，再利用 gdb 编译 bomb 文件（图 1）；在主函数入口打断点，即输入指令 break main，运行 run（图 2），显示在第 37 行第一个断点处停止；再利用 disas 查看汇编代码，我么能够看到<+154> callq <phase_1>指令（图 3）；

```
2021210967@bupt1:~/bomb33$ vi bomb.c
2021210967@bupt1:~/bomb33$ 2021210967@bupt1:~/bomb33$ gdb bomb
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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-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...
```

```
(gdb) break main
Breakpoint 1 at 0x400df6: file bomb.c, line 37.
(gdb) run
Starting program: /students/2021210967/bomb33/bomb

Breakpoint 1, main (argc=1, argv=0x7fffffffeba8) at bomb.c:37
warning: Source file is more recent than executable.
37      {
```

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000000000400df6 <+0>:     push   %rbx
   0x0000000000400df7 <+1>:     cmp    $0x1,%edi
   0x0000000000400dfa <+4>:     jne    0x400e0c <main+22>
   0x0000000000400dfc <+6>:     mov    0x20398d(%rip),%rax        # 0x604790 <stdin@@GLIBC_2.2.5>
   0x0000000000400e03 <+13>:    mov    %rax,0x2039a6(%rip)        # 0x6047b0 <infile>
   0x0000000000400e0a <+20>:    jmp    0x400e6f <main+121>
   0x0000000000400e0c <+22>:    mov    %rsi,%rbx
   0x0000000000400e0f <+25>:    cmp    $0x2,%edi
   0x0000000000400e12 <+28>:    jne    0x400e4e <main+88>
   0x0000000000400e14 <+30>:    mov    0x8(%rsi),%rdi
   0x0000000000400e18 <+34>:    mov    $0x4025a4,%esi
   0x0000000000400e1d <+39>:    callq  0x400c60 <fopen@plt>
   0x0000000000400e22 <+44>:    mov    %rax,0x203987(%rip)        # 0x6047b0 <infile>
   0x0000000000400e29 <+51>:    test   %rax,%rax
   0x0000000000400e2c <+54>:    jne    0x400e6f <main+121>
   0x0000000000400e2e <+56>:    mov    0x8(%rbx),%rcx
   0x0000000000400e32 <+60>:    mov    (%rbx),%rdx
   0x0000000000400e35 <+63>:    mov    $0x4025a6,%esi
   0x0000000000400e3a <+68>:    mov    $0x1,%edi
   0x0000000000400e3f <+73>:    callq  0x400c50 <__printf_chk@plt>
   0x0000000000400e44 <+78>:    mov    $0x8,%edi
   0x0000000000400e49 <+83>:    callq  0x400c80 <exit@plt>
   0x0000000000400e4e <+88>:    mov    (%rsi),%rdx
   0x0000000000400e51 <+91>:    mov    $0x4025c3,%esi
   0x0000000000400e56 <+96>:    mov    $0x1,%edi
   0x0000000000400e5b <+101>:   mov    $0x0,%eax
   0x0000000000400e60 <+106>:   callq  0x400c50 <__printf_chk@plt>
   0x0000000000400e65 <+111>:   mov    $0x8,%edi
   0x0000000000400e6a <+116>:   callq  0x400c80 <exit@plt>
   0x0000000000400e6f <+121>:   callq  0x4014a6 <initialize_bomb>
   0x0000000000400e74 <+126>:   mov    $0x402628,%edi
   0x0000000000400e79 <+131>:   callq  0x400b70 <puts@plt>
   0x0000000000400e7e <+136>:   mov    $0x402668,%edi
   0x0000000000400e83 <+141>:   callq  0x400b70 <puts@plt>
   0x0000000000400e88 <+146>:   callq  0x401788 <read_line>
   0x0000000000400e8d <+151>:   mov    %rax,%rdi
   0x0000000000400e90 <+154>:   callq  0x400f2d <phase_1>
   0x0000000000400e95 <+159>:   callq  0x4018ae <phase_defused>
   0x0000000000400e9a <+164>:   mov    $0x402698,%edi
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400e9f <+169>:   callq  0x400b70 <puts@plt>
   0x0000000000400ea4 <+174>:   callq  0x401788 <read_line>
```

3. 在 phase_1 处再次打断点，继续执行，这时候出现炸弹提示，我们输入 test 检查当前运行停止在了第二个断点 phase_1 处，继续运行；

```
(gdb) break phase_1
Breakpoint 2 at 0x400f2d
(gdb) c
Continuing.
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
test

Breakpoint 2, 0x0000000000400f2d in phase_1 ()
(gdb) si
0x0000000000400f31 in phase_1 ()
```

4. 再次利用 disas 反汇编，查看 phase_1 汇编代码，当前指令行为<+4> mov $0x4026f0,%esi，在经历函数 strings_not_equal 之后,<+14>检验寄存器%eax 中的值是否等于 0,如果等于 0,就安全退出 bomb，如果不等于 0，则炸弹爆发（图 1）；我们继续运行程序，直到 strings_not_equal；

```
(gdb) disas
Dump of assembler code for function phase_1:
   0x0000000000400f2d <+0>:     sub    $0x8,%rsp
=> 0x0000000000400f31 <+4>:     mov    $0x4026f0,%esi
   0x0000000000400f36 <+9>:     callq  0x40143f <strings_not_equal>
   0x0000000000400f3b <+14>:    test   %eax,%eax
   0x0000000000400f3d <+16>:    je     0x400f44 <phase_1+23>
   0x0000000000400f3f <+18>:    callq  0x401713 <explode_bomb>
   0x0000000000400f44 <+23>:    add    $0x8,%rsp
   0x0000000000400f48 <+27>:    retq
End of assembler dump.
(gdb) si
0x0000000000400f36 in phase_1 ()
(gdb) si
0x000000000040143f in strings_not_equal ()
```

5. 接下来查看寄存器%esi 中的值，退出当前程序调试，重新在 phase_1 处打断点，并运行，bomb1 实验完成。Have a nice day!

```
(gdb) x/s $esi
0x4026f0:       "In 2012, BUPT officially started the construction of new ShaHe campus."
(gdb) kill
Kill the program being debugged? (y or n) y
[Inferior 1 (process 1308636) killed]
(gdb) delete
Delete all breakpoints? (y or n) y
(gdb) break phase_1
Breakpoint 3 at 0x400f2d
(gdb) run
Starting program: /students/2021210967/bomb33/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
```

● 实验二

1. 首先我们使用 vi 软件查看“bomb.c”源代码文件，找到第二关的 C 代码段，如图所示：

```
    /* The second phase is harder.  No one will ever figure out
     * how to defuse this... */
    input = read_line();
    phase_2(input);
    phase_defused();
    printf("That's number 2.  Keep going!\n");

    /* I guess this is too easy so far.  Some more complex code will
     * confuse people. */
```

2. 再查看 phase_2 函数的反汇编代码，如下所示，图片下方使对汇编指令的分析；

```
(gdb) b phase_2
Breakpoint 1 at 0x400f49
(gdb) c
The program is not being run.
(gdb) run
Starting program: /students/2021210967/bomb33/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
In 2012, BUPT officially started the construction of new ShaHe campus.
Phase 1 defused. How about the next one?
test

Breakpoint 1, 0x0000000000400f49 in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x0000000000400f49 <+0>:     push   %rbp
   0x0000000000400f4a <+1>:     push   %rbx
   0x0000000000400f4b <+2>:     sub    $0x28,%rsp
   0x0000000000400f4f <+6>:     mov    %fs:0x28,%rax
   0x0000000000400f58 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400f5d <+20>:    xor    %eax,%eax
   0x0000000000400f5f <+22>:    mov    %rsp,%rsi
   0x0000000000400f62 <+25>:    callq  0x401749 <read_six_numbers>
   0x0000000000400f67 <+30>:    cmpl   $0x0,(%rsp)
   0x0000000000400f6b <+34>:    jns    0x400f72 <phase_2+41>
   0x0000000000400f6d <+36>:    callq  0x401713 <explode_bomb>
   0x0000000000400f72 <+41>:    mov    %rsp,%rbp
   0x0000000000400f75 <+44>:    mov    $0x1,%ebx
   0x0000000000400f7a <+49>:    mov    %ebx,%eax
   0x0000000000400f7c <+51>:    add    0x0(%rbp),%eax
   0x0000000000400f7f <+54>:    cmp    %eax,0x4(%rbp)
   0x0000000000400f82 <+57>:    je     0x400f89 <phase_2+64>
   0x0000000000400f84 <+59>:    callq  0x401713 <explode_bomb>
   0x0000000000400f89 <+64>:    add    $0x1,%ebx
   0x0000000000400f8c <+67>:    add    $0x4,%rbp
   0x0000000000400f90 <+71>:    cmp    $0x6,%ebx
   0x0000000000400f93 <+74>:    jne    0x400f7a <phase_2+49>
   0x0000000000400f95 <+76>:    mov    0x18(%rsp),%rax
   0x0000000000400f9a <+81>:    xor    %fs:0x28,%rax
   0x0000000000400fa3 <+90>:    je     0x400faa <phase_2+97>
(gdb)
   0x0000000000400faa <+97>:    add    $0x28,%rsp
   0x0000000000400fae <+101>:   pop    %rbx
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400faf <+102>:   pop    %rbp
   0x0000000000400fb0 <+103>:   retq
End of assembler dump.
```

Dump of assembler code for function phase_2:

=> 0x0000000000400f49 <+0>: push %rbp

0x0000000000400f4a <+1>: push %rbx

0x0000000000400f4b <+2>: sub $0x28,%rsp//0x28=40=4*10，栈指针下移 10 个字节

0x0000000000400f4f <+6>: mov %fs:0x28,%rax

0x0000000000400f58 <+15>: mov %rax,0x18(%rsp)

0x0000000000400f5d <+20>: xor %eax,%eax//%eax 为 0

0x0000000000400f5f <+22>: mov %rsp,%rsi

0x0000000000400f62 <+25>: callq 0x401749 <read_six_numbers>//这里可知须输入六个数字；

0x0000000000400f67 <+30>: cmpl $0x0,(%rsp)//比较栈顶元素与 0 的大小

0x0000000000400f6b <+34>: jns 0x400f72 <phase_2+41>//比 0 大，跳转

0x0000000000400f6d <+36>: callq 0x401713 <explode_bomb>

0x0000000000400f72 <+41>: mov %rsp,%rbp

0x0000000000400f75 <+44>: mov $0x1,%ebx//循环指数 i 的初始值为 1；

0x0000000000400f7a <+49>: mov %ebx,%eax//循环使 i 加一；

0x0000000000400f7c <+51>: add 0x0(%rbp),%eax//%eax 内存 i+栈顶元素【上一个数】

0x0000000000400f7f <+54>: cmp %eax,0x4(%rbp)//比较 i+前一个数与当前栈顶元素；

0x0000000000400f82 <+57>: je 0x400f89 <phase_2+64>//如果相等，跳出；

0x0000000000400f84 <+59>: callq 0x401713 <explode_bomb>//如果不等，触发炸弹；

0x0000000000400f89 <+64>: add $0x1,%ebx//ebx 加一，相当于循环指数 i 加一；

0x0000000000400f8c <+67>: add $0x4,%rbp//栈指针向上移动

```
0x0000000000400f90 <+71>:    cmp    $0x6,%ebx//i<6
0x0000000000400f93 <+74>:    jne    0x400f7a <phase_2+49>//从这里可以看出来是一个循环；
0x0000000000400f95 <+76>:    mov    0x18(%rsp),%rax
0x0000000000400f9a <+81>:    xor    %fs:0x28,%rax
0x0000000000400fa3 <+90>:    je     0x400faa <phase_2+97>
0x0000000000400fa5 <+92>:    callq  0x400b90 <__stack_chk_fail@plt>
0x0000000000400faa <+97>:    add    $0x28,%rsp
0x0000000000400fae <+101>:   pop    %rbx
```

3. 分析汇编代码，可知在<+25>行指令中，我们输入的格式是六个数字；在<+54> 和 <+59>行指令中可以分析出，对于输入的六个数，如果下一个数等于它前一个数加当前循环 i，则成功通关；设置第一个数字为 0，则答案为 0 1 3 6 10 15，这是一个相邻元素的差等差的数列；

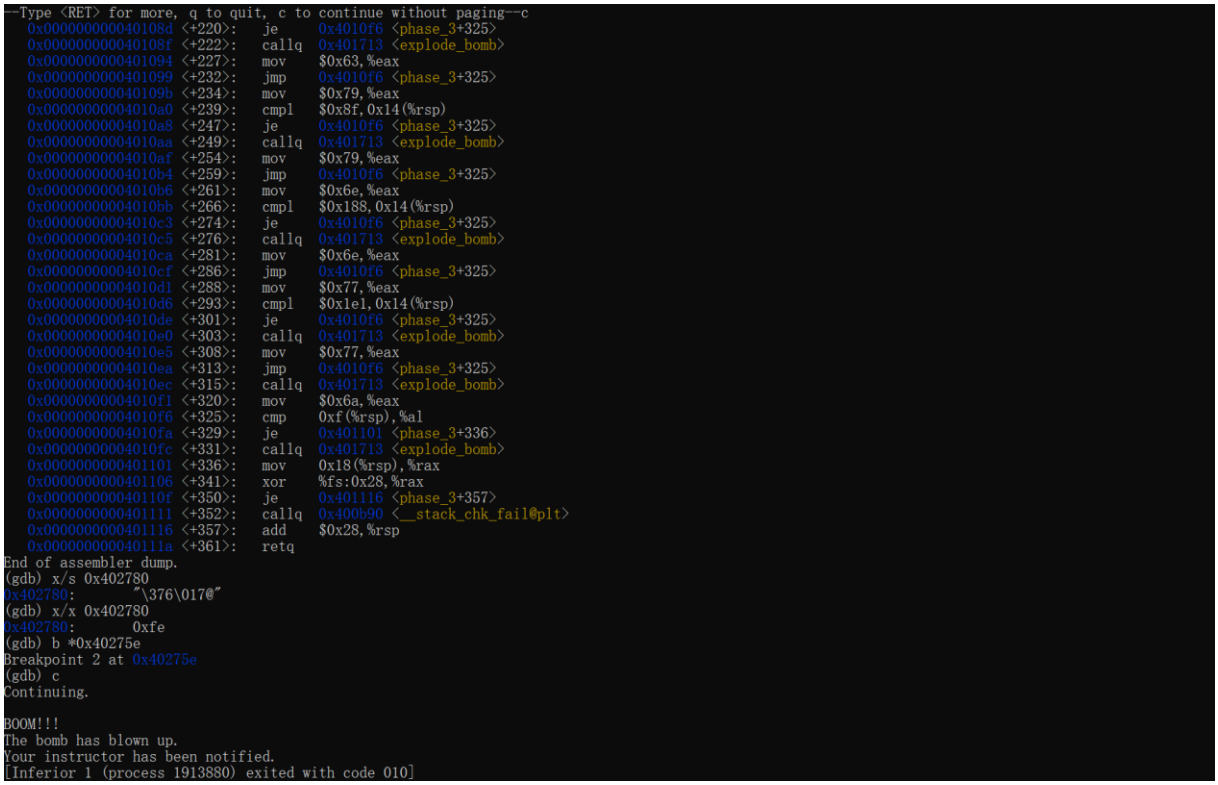
```
(gdb) b phase_2
Breakpoint 1 at 0x400f49
(gdb) run
Starting program: /students/2021210967/bomb33/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
In 2012, BUPT officially started the construction of new ShaHe campus.
Phase 1 defused. How about the next one?
0 1 3 6 10 15

Breakpoint 1, 0x0000000000400f49 in phase_2 ()
(gdb) c
Continuing.
That's number 2.  Keep going!
```

● 实验三

1. 依据前两个实验的步骤，再次查看第三个函数的汇编代码，第一次查看地址的时候查看地址错误，不知道要查看左侧指令地址，于是 BOOM 了……，改正之后才知道需要输入的格式是 %d %c %d，即两个数字加一个字符；
2. 继续查看汇编代码，确实有一些长…；图后面是具体的汇编指令解题分析；

```
(gdb) disas
Dump of assembler code for function phase_3:
=> 0x0000000000400fb1 <+0>:     sub    $0x28,%rsp
   0x0000000000400fb5 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fbe <+13>:    mov    %rax,0x18(%rsp)
   0x0000000000400fc3 <+18>:    xor    %eax,%eax
   0x0000000000400fc5 <+20>:    lea    0x14(%rsp),%r8
   0x0000000000400fca <+25>:    lea    0xf(%rsp),%rcx
   0x0000000000400fcf <+30>:    lea    0x10(%rsp),%rdx
   0x0000000000400fd4 <+35>:    mov    $0x40275e,%esi
   0x0000000000400fd9 <+40>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000400fde <+45>:    cmp    $0x2,%eax
   0x0000000000400fe1 <+48>:    jg     0x400fe8 <phase_3+55>
   0x0000000000400fe3 <+50>:    callq  0x401713 <explode_bomb>
   0x0000000000400fe8 <+55>:    cmpl   $0x7,0x10(%rsp)
   0x0000000000400fed <+60>:    ja     0x4010ec <phase_3+315>
   0x0000000000400ff3 <+66>:    mov    0x10(%rsp),%eax
   0x0000000000400ff7 <+70>:    jmpq   *0x402780(,%rax,8)
   0x0000000000400ffe <+77>:    mov    $0x6b,%eax
   0x0000000000401003 <+82>:    cmpl   $0x40,0x14(%rsp)
   0x0000000000401008 <+87>:    je     0x4010f6 <phase_3+325>
   0x000000000040100e <+93>:    callq  0x401713 <explode_bomb>
   0x0000000000401013 <+98>:    mov    $0x6b,%eax
   0x0000000000401018 <+103>:   jmpq   0x4010f6 <phase_3+325>
   0x000000000040101d <+108>:   mov    $0x6c,%eax
   0x0000000000401022 <+113>:   cmpl   $0xa0,0x14(%rsp)
   0x000000000040102a <+121>:   je     0x4010f6 <phase_3+325>
   0x0000000000401030 <+127>:   callq  0x401713 <explode_bomb>
   0x0000000000401035 <+132>:   mov    $0x6c,%eax
   0x000000000040103a <+137>:   jmpq   0x4010f6 <phase_3+325>
   0x000000000040103f <+142>:   mov    $0x70,%eax
   0x0000000000401044 <+147>:   cmpl   $0x122,0x14(%rsp)
   0x000000000040104c <+155>:   je     0x4010f6 <phase_3+325>
   0x0000000000401052 <+161>:   callq  0x401713 <explode_bomb>
   0x0000000000401057 <+166>:   mov    $0x70,%eax
   0x000000000040105c <+171>:   jmpq   0x4010f6 <phase_3+325>
   0x0000000000401061 <+176>:   mov    $0x65,%eax
   0x0000000000401066 <+181>:   cmpl   $0xf3,0x14(%rsp)
   0x000000000040106e <+189>:   je     0x4010f6 <phase_3+325>
   0x0000000000401074 <+195>:   callq  0x401713 <explode_bomb>
   0x0000000000401079 <+200>:   mov    $0x65,%eax
(gdb) _
   0x0000000000401080 <+207>:   mov    $0x63,%eax
   0x0000000000401085 <+212>:   cmpl   $0x135,0x14(%rsp)
--Type <RET> for more, q to quit, c to continue without paging--c
```



Dump of assembler code for function phase_3:

```
=> 0x0000000000400fb1 <+0>:     sub    $0x28,%rsp
   0x0000000000400fb5 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fbe <+13>:    mov    %rax,0x18(%rsp)
   0x0000000000400fc3 <+18>:    xor    %eax,%eax
   0x0000000000400fc5 <+20>:    lea    0x14(%rsp),%r8
   0x0000000000400fca <+25>:    lea    0xf(%rsp),%rcx
   0x0000000000400fcf <+30>:    lea    0x10(%rsp),%rdx
   0x0000000000400fd4 <+35>:    mov    $0x40275e,%esi
   0x0000000000400fd9 <+40>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000400fde <+45>:    cmp    $0x2,%eax
   0x0000000000400fe1 <+48>:    jg     0x400fe8 <phase_3+55>
   0x0000000000400fe3 <+50>:    callq  0x401713 <explode_bomb>
   0x0000000000400fe8 <+55>:    cmpl   $0x7,0x10(%rsp)//比较第一个输入的数值与 7 的大小；
   0x0000000000400fed <+60>:    ja     0x4010ec <phase_3+315>//如果大于 7，跳到炸弹函数；
   0x0000000000400ff3 <+66>:    mov    0x10(%rsp),%eax
   0x0000000000400ff7 <+70>:    jmpq   *0x402780(,%rax,8)
   0x0000000000400ffe <+77>:    mov    $0x6b,%eax
   0x0000000000401003 <+82>:    cmpl   $0x40,0x14(%rsp)//将栈顶指针与 64 作比较；
   0x0000000000401008 <+87>:    je     0x4010f6 <phase_3+325>//如果相等，跳到<phase_3+325>行
   ……【省略】
   0x00000000004010f6 <+325>:   cmp    0xf(%rsp),%al
   0x00000000004010fa <+329>:   je     0x401101 <phase_3+336>
   0x00000000004010fc <+331>:   callq  0x401713 <explode_bomb>
```

```
0x0000000000401101 <+336>:    mov    0x18(%rsp),%rax
0x0000000000401106 <+341>:    xor    %fs:0x28,%rax
0x000000000040110f <+350>:    je     0x401116 <phase_3+357>
0x0000000000401111 <+352>:    callq  0x400b90 <__stack_chk_fail@plt>
0x0000000000401116 <+357>:    add    $0x28,%rsp
0x000000000040111a <+361>:    retq
```

3. 根据第<+55>行指令，可知我们输入的第一个数值必须小于 7；我选择 0；根据第<+77>个指令，证明%eax 内赋值为 0x6b 即十进制数 107，对应第二个输入字符为“k”；于是根据<+82>行指令计算出第三个输入为 64；最终答案为 0 k 64，解题成功！

```
That's number 2.  Keep going!
0 k 64
Halfway there!
```

● 实验四

1. 重复查看实验四的汇编代码，此时看到了出现了新的函数 func4，再查看 func4 的汇编代码；

```
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x0000000000401159 <+0>:     sub    $0x18,%rsp
   0x000000000040115d <+4>:     mov    %fs:0x28,%rax
   0x0000000000401166 <+13>:    mov    %rax,0x8(%rsp)
   0x000000000040116b <+18>:    xor    %eax,%eax
   0x000000000040116d <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000401172 <+25>:    mov    %rsp,%rdx
   0x0000000000401175 <+28>:    mov    $0x402a4d,%esi
   0x000000000040117a <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x000000000040117f <+38>:    cmp    $0x2,%eax
   0x0000000000401182 <+41>:    jne    0x40118a <phase_4+49>
   0x0000000000401184 <+43>:    cmpl   $0xe,(%rsp)
   0x0000000000401188 <+47>:    jbe    0x40118f <phase_4+54>
   0x000000000040118a <+49>:    callq  0x401713 <explode_bomb>
   0x000000000040118f <+54>:    mov    $0xe,%edx
   0x0000000000401194 <+59>:    mov    $0x0,%esi
   0x0000000000401199 <+64>:    mov    (%rsp),%edi
   0x000000000040119c <+67>:    callq  0x40111b <func4>
   0x00000000004011a1 <+72>:    cmp    $0x4,%eax
   0x00000000004011a4 <+75>:    jne    0x4011ad <phase_4+84>
   0x00000000004011a6 <+77>:    cmpl   $0x4,0x4(%rsp)
   0x00000000004011ab <+82>:    je     0x4011b2 <phase_4+89>
   0x00000000004011ad <+84>:    callq  0x401713 <explode_bomb>
   0x00000000004011b2 <+89>:    mov    0x8(%rsp),%rax
   0x00000000004011b7 <+94>:    xor    %fs:0x28,%rax
   0x00000000004011c0 <+103>:   je     0x4011c7 <phase_4+110>
   0x00000000004011c2 <+105>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x00000000004011c7 <+110>:   add    $0x18,%rsp
(gdb)
--Type <RET> for more, q to quit, c to continue without paging--c
End of assembler dump.
(gdb)
```

2. 对于 func4 的汇编代码，可以发现，第四题在 func4 函数中调用了 func4 函数，明显是递归，因为汇编代码较短，直接写了程序，如下图图三；
3. 根据查看地址 0x402a4d 中的数值，可以得到输入为%d %d 即两个整数；根据第<+38>行指令，可以知道第一个输入的数值必须为 2，否则触发炸弹；而此值，在下面也被 func4 函数调用了，因此，第一个输入的值决定了第二个输入的值，即最后判断出递归的条件；查看代码结果，函数返回值为 4【在<+72>行指令中也可以推断出来】，最终答案为 2 4；

```
(gdb) disas func4
Dump of assembler code for function func4:tinue without paging--c
   0x000000000040111b <+0>:     sub    $0x8,%rsp
   0x000000000040111f <+4>:     mov    %edx,%eax
   0x0000000000401121 <+6>:     sub    %esi,%eax
   0x0000000000401123 <+8>:     mov    %eax,%ecx
   0x0000000000401125 <+10>:    shr    $0x1f,%ecx
   0x0000000000401128 <+13>:    add    %ecx,%eax
   0x000000000040112a <+15>:    sar    %eax
   0x000000000040112c <+17>:    lea    (%rax,%rsi,1),%ecx
   0x000000000040112f <+20>:    cmp    %edi,%ecx
   0x0000000000401131 <+22>:    jle    0x40113f <func4+36>
   0x0000000000401133 <+24>:    lea    -0x1(%rcx),%edx
   0x0000000000401136 <+27>:    callq  0x40111b <func4>
   0x000000000040113b <+32>:    add    %eax,%eax
   0x000000000040113d <+34>:    jmp    0x401154 <func4+57>
   0x000000000040113f <+36>:    mov    $0x0,%eax
   0x0000000000401144 <+41>:    cmp    %edi,%ecx
   0x0000000000401146 <+43>:    jge    0x401154 <func4+57>
   0x0000000000401148 <+45>:    lea    0x1(%rcx),%esi
   0x000000000040114b <+48>:    callq  0x40111b <func4>
   0x0000000000401150 <+53>:    lea    0x1(%rax,%rax,1),%eax
   0x0000000000401154 <+57>:    add    $0x8,%rsp
   0x0000000000401158 <+61>:    retq
End of assembler dump.
(gdb)
```

```cpp
#include<stdio.h>
#include<iostream>
using namespace std;

int edx, esi, eax, ecx;

int func4(int edi) {
    eax = edx;
    eax-=esi;
    ecx = eax;
    ecx >>= 0x1f;
    eax +=ecx;
    eax >>= 1;
    ecx = esi + eax;
    if (ecx <= edi){
        eax = 0;
        if (ecx >= edi);
        else{
            esi = ecx + 1;
            eax = 2 * func4(edi) + 1;
        }
    }
    else{
        edx = ecx - 1;
        eax = func4(edi);
        eax += eax;
    }
    return eax;
}

int main() {
    int edi;
    for (edi = 0; edi <= 13; edi++){
        edx = 14;
        esi = 0;
        cout << "终止条件:" << edi+1 << " 函数返回值:" << func4(edi+1) << endl;
    }

    return 0;
}

```

```
C:\Users\hua'wei\Desktop\1.exe
终止条件:1 函数返回值:0
终止条件:2 函数返回值:4
终止条件:3 函数返回值:0
终止条件:4 函数返回值:2
终止条件:5 函数返回值:2
终止条件:6 函数返回值:6
终止条件:7 函数返回值:0
终止条件:8 函数返回值:1
终止条件:9 函数返回值:1
终止条件:10 函数返回值:5
终止条件:11 函数返回值:1
终止条件:12 函数返回值:3
终止条件:13 函数返回值:3
终止条件:14 函数返回值:7

--------------------------------
Process exited after 0.5101 seconds with return value 0
请按任意键继续. . .
```

● 实验五

1. 做到这里已经对拆解炸弹有了一定思路的考究，逐行分析指令对于有的题似乎并不是最好的方法；
2. 首先找到跳出 explode 函数的条件，再找到所有汇编指令里有 comp 对比有关的语句，确定题目源代码的整体结构；

```
Dump of assembler code for function phase_5:
=> 0x00000000004011cc <+0>:     push   %rbx
   0x00000000004011cd <+1>:     mov    %rdi,%rbx
   0x00000000004011d0 <+4>:     callq  0x401421 <string_length>
   0x00000000004011d5 <+9>:     cmp    $0x6,%eax
   0x00000000004011d8 <+12>:    je     0x4011df <phase_5+19>
   0x00000000004011da <+14>:    callq  0x401713 <explode_bomb>
   0x00000000004011df <+19>:    mov    %rbx,%rax
   0x00000000004011e2 <+22>:    lea    0x6(%rbx),%rdi
   0x00000000004011e6 <+26>:    mov    $0x0,%ecx
   0x00000000004011eb <+31>:    movzbl (%rax),%edx
   0x00000000004011ee <+34>:    and    $0xf,%edx
   0x00000000004011f1 <+37>:    add    0x4027c0(,%rdx,4),%ecx
   0x00000000004011f8 <+44>:    add    $0x1,%rax
   0x00000000004011fc <+48>:    cmp    %rdi,%rax
   0x00000000004011ff <+51>:    jne    0x4011eb <phase_5+31>
   0x0000000000401201 <+53>:    cmp    $0x1e,%ecx
   0x0000000000401204 <+56>:    je     0x40120b <phase_5+63>
   0x0000000000401206 <+58>:    callq  0x401713 <explode_bomb>
   0x000000000040120b <+63>:    pop    %rbx
   0x000000000040120c <+64>:    retq
End of assembler dump.
(gdb)
```

```
(gdb) disas
Dump of assembler code for function phase_5:
=> 0x00000000004011cc <+0>:     push   %rbx
   0x00000000004011cd <+1>:     mov    %rdi,%rbx
   0x00000000004011d0 <+4>:     callq  0x401421 <string_length>
   0x00000000004011d5 <+9>:     cmp    $0x6,%eax
   0x00000000004011d8 <+12>:    je     0x4011df <phase_5+19>
   0x00000000004011da <+14>:    callq  0x401713 <explode_bomb>
   0x00000000004011df <+19>:    mov    %rbx,%rax
   0x00000000004011e2 <+22>:    lea    0x6(%rbx),%rdi
   0x00000000004011e6 <+26>:    mov    $0x0,%ecx
   0x00000000004011eb <+31>:    movzbl (%rax),%edx
   0x00000000004011ee <+34>:    and    $0xf,%edx
   0x00000000004011f1 <+37>:    add    0x4027c0(,%rdx,4),%ecx
   0x00000000004011f8 <+44>:    add    $0x1,%rax
   0x00000000004011fc <+48>:    cmp    %rdi,%rax
   0x00000000004011ff <+51>:    jne    0x4011eb <phase_5+31>
   0x0000000000401201 <+53>:    cmp    $0x1e,%ecx
   0x0000000000401204 <+56>:    je     0x40120b <phase_5+63>
   0x0000000000401206 <+58>:    callq  0x401713 <explode_bomb>
   0x000000000040120b <+63>:    pop    %rbx
   0x000000000040120c <+64>:    retq
End of assembler dump.
```

3. 输入：结合第<+4>、<+9>、<+12>个指令可知，输入为六个字符组成的字符串；
4. 在第<+53>行，可知跳出 explode 的条件是%ecx 所存的数值与 30 相等；
5. 由【<+37>: add 0x4027c0(,%rdx,4),%ecx】此指令可知道，%ecx 最终是通过累加得来，而%ecx 初值为 0，经历六次循环之后得到了 30。而其累加的值，即取了地址 0x4027c0 中的数值，我们查看这个地址，发现此数组中有 16 个数值，为达到 6 此循环结果为 30，我们选择最简单的情况即 6 个 5 相加；

```
(gdb) x/16dw 0x4027c0
0x4027c0 <array.3602>:  2       10      6       1
0x4027d0 <array.3602+16>:       12      16      9       3
0x4027e0 <array.3602+32>:       4       7       14      5
0x4027f0 <array.3602+48>:       11      8       15      13
```

6. 那么怎么得到最终我们需要输入的字符串的 6 个字符呢？对于<+34>: and $0xf,%edx 此指令来说，%edx 最终寄存的是初始%rdi 中的值，因此，and 操作后，我们得到了输入字符对应的 ANSCII 码，于是我们得到了 107，对应字符 k；

   最终结果输入为 6 个相同字符 kkkkkk；

```
(gdb) k
Kill the program being debugged? (y or n) y
[Inferior 1 (process 3043182) killed]
(gdb) del
Delete all breakpoints? (y or n) y
(gdb) b explode_bomb
Breakpoint 3 at 0x401713
(gdb) r
Starting program: /students/2021210967/bomb33/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
In 2012, BUPT officially started the construction of new ShaHe campus.
Phase 1 defused. How about the next one?
0 1 3 6 10 15
That's number 2.  Keep going!
0 k 64
Halfway there!
2 4
So you got that one.  Try this one.
kkkkkk
Good work!  On to the next...
```

五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

这次的拆解二进制炸弹闯关实验，非常独特，没有枯燥乏味的一味的照搬照抄，而是把知识应用到了拆解的过程。拆解二进制炸弹的过程锻炼了我们读汇编代码的能力，考验了我们对一些汇编代码的重新认识，以及通过实际题目了解到了汇编代码的真实使用方法，感觉自己与机器又近了一步。

但是对于我这样一个对汇编代码掌握不太熟练，并且不太了解这个 bomb 实验以至于畏手畏脚一直不敢大手大脚查看自己想查看的地址的人，（实际上是因为我在无意之中炸了三次），然后到第三个实验的时候就将 disas 汇编代码复制，退出 powershell 然后单独阅读翻译；直到第四个实验，别人教给了一个方法是在 explode_bomb 函数行打断点，b explode_bomb，才没有那么谨慎的进行接下来的实验；

我给自己的总结不足的地方就是，对 gdb 编译器的使用还没有那么熟练，但是经过 lab1 和 lab2 之后，也渐渐理解了汇编编译器的好用之处，更加深入理解了这些语言的逻辑以及使用目的，还是很奇妙的一次体验；

完成实验后，我总结了五个关卡分别考查的内容：

关卡一：查看字符串，第一个实验确实相对比较简单，但由于我的语句很长，还没有复制粘贴功能，第一次我也忘记了存答案入文件，以至于我每次都打一堆字……

关卡二：输入一个有规律的数列：这个题我了解到的有三种不同的数列组合，等比、斐波那契、还有我的差等差数列；汇编代码还是比较容易读懂的，前两个实验独立完成，问题不大；

关卡三：通过汇编代码，猜测第三题考察 switch 语句，除了汇编代码特别特别长之外…但其实中间一大块都没有用处；根据前后的 cmp 关系，大致能够接出输入输出；当然了，有的同学输入为两个整数，而我是两个整数夹杂一个字符，每日一问为啥脑袋不太灵光的我抽到了比较抽象的题目（不是）；但其实本质上题目是一样的，只是多了 ANSCII 码的转换；

关卡四：递归对于我来说一直都很抽象，尤其是反复调用递归函数，一层一层的调用，由于代码比较短，直接写了相关的代码，最终的答案比较一目了然；

关卡五：第五个实验关键就在于指针数组寻址问题，其关键点在于寻找到数组第一个元素【题中明显可以找到】的地址和该数组的长度，通过查看数组内数字的具体数据，可以最终凑成我们需要的答案；当然，这个题目也用到了 ANSCII 码的转换。

总的来说，Dt.Evil 也不算真的 evil，反而给予了一般实验课没有的趣味性，当然，BOOM!!! 的时候也很害怕，总之，希望自己能在后半个学期的实验中逐渐地真正地深入了解计算机系统吧！