课程编号 1502760001-07

题目类型 实验 3

得分	教师签名	批改日期
	冯禹洪	

深 圳 大 学 实 验 报 告

课程名称：计算机系统(2)

实验项目名称：逆向工程实验

学院：计算机与软件学院

专业：软件工程（腾班）

指导教师：冯禹洪

报告人：黄亮铭 学号：2022155028 班级：腾班

实验时间：2024 年 4 月 27 日至 5 月 10 日

实验报告提交时间：2024 年 5 月 10 日

教务处制

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

一、 实验目标与要求：

1. 理解程序（控制语句、函数、返回值、堆栈结构）是如何运行的
2. 掌握 GDB 调试工具和 objdump 反汇编工具

二、实验环境：

1. 计算机（Intel CPU）
2. Linux64 位操作系统（Ubuntu 17）
3. GDB 调试工具
4. objdump 反汇编工具

三、实验方法与步骤：

本实验设计为一个黑客拆解二进制炸弹的游戏。我们仅给黑客（同学）提供一个二进制可执行文件 bomb_64 和主函数所在的源程序 bomb_64.c，不提供每个关卡的源代码。程序运行中有 6 个关卡（6 个 phase），每个关卡需要用户输入正确的字符串或数字才能通关，否则会引爆炸弹（打印出一条错误信息，并导致评分下降）!

要求同学运用 **GDB 调试工具和 objdump 反汇编工具**，通过分析汇编代码，找到在每个 phase 程序段中，引导程序跳转到“explode_bomb”程序段的地方，并分析其成功跳转的条件，以此为突破口寻找应该在命令行输入何种字符串来通关。

本实验需解决 Phase_1(**15 分**)、Phase_2(**15 分**)、Phase_3(**15 分**)、Phase_4(**15 分**)、Phase_5(**15 分**)、Phase_6(**10 分**)。通过**截图+文字**的形式把实验过程写在实验报告上，最后并撰写**实验结论与心得(15 分**)。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

四、实验过程及内容：

前期准备

1. 使用命令$sudo\ chmod\ a+rw\ Result.txt$为$Result.txt$提供读写权限。
2. 使用命令$objdump\ -d\ bomb_64\ >\ Result.txt$ 对 bomb 文件进行反汇编，并将结果输出到 1.txt 中。
3. 使用$VsCode$打开$1.txt$文件，并定位到 main 函数中。
4. 阅读 main 函数，然后跳转到相应的关卡。

第一关

阅读 *phase_1* 的代码。我发现程序首先申请栈空间，然后将 0x401af8 处的内容存储到寄存集*%esi* 中。接下来调用函数 *strings_not_equal*，合理猜测这个函数的作用是判断字符串和系统给定的字符串是否相等，相等返回 0，否则返回其他。因为*%eai* 是存储函数返回值的寄存器，0x400e80 处判断返回值是否为 0，如果为 0 则跳转。综上所述，我们需要输入一个字符串，使其和 0x401af8 处的内容相等。

```
0000000000400e70 <phase_1>:
  400e70: 48 83 ec 08           sub    $0x8,%rsp
  400e74: be f8 1a 40 00        mov    $0x401af8,%esi
  400e79: e8 bf 03 00 00        call   40123d <strings_not_equal>
  400e7e: 85 c0                 test   %eax,%eax
  400e80: 74 05                 je     400e87 <phase_1+0x17>
  400e82: e8 b6 07 00 00        call   40163d <explode_bomb>
  400e87: 48 83 c4 08           add    $0x8,%rsp
  400e8b: c3                    ret
```

我们进入 GDB 调试获取 0x401af8 处的内容。

```
Reading symbols from bomb_64...
(gdb) p (char*)0x401af8
$1 = 0x401af8 "Science isn't about why, it's about why not?"
(gdb)
```

发现是上图所示的内容，所以在测试阶段输入上述内容即可通过当前关卡。

第二关

阅读 *phase_2* 的代码。我发现程序（0x400e8c-0x400ea0）首先申请栈空间将需要用到的寄存器的值保存起来并在函数结束时还原（0x400ee0-0x400ef4）。接下来将栈指针赋给*%rsi*，作为参数传入 *read_six_numbers*。根据函数名可知，该函数的作用是读取 6 个数字，这 6 个数字需要符合一定规则。

继续阅读代码。*read_six_numbers* 函数调用结束后，程序将*%rsp* 将赋值给*%rsi* 作为迭代指针。然后将*%rsp+12* 处的值赋给*%r13* 作为循环结束的判断条件，清空*%r12* 将其作为累加器。

接下来程序进入循环（0x400eba-0x400ed4），将%rbp 赋值给%rbx，将%rbp+12 指向的内存地址的内容赋值给%eax，然后比较%eax 的数据和%rbp 指向的内存地址的数据是否相

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

等。如果不相等则触发炸弹，如果相等则跳过触发炸弹的函数。再将%rbx 指向的内存地址的数据累加到%r12。最后迭代指针加 4 即指向下一个数据，然后判断%r13 和%rbp 是否相等即是否遍历到第 4 个数据，如是则退出循环，否则继续循环。
然后判断累加器%r12d 是否等于 0，如果等于 0，则触发炸弹，否则不触发。最后执行上面提到的还原操作。

```
0000000000400e8c <phase_2>:
  400e8c: 48 89 5c 24 e0        mov    %rbx,-0x20(%rsp)
  400e91: 48 89 6c 24 e8        mov    %rbp,-0x18(%rsp)
  400e96: 4c 89 64 24 f0        mov    %r12,-0x10(%rsp)
  400e9b: 4c 89 6c 24 f8        mov    %r13,-0x8(%rsp)
  400ea0: 48 83 ec 48           sub    $0x48,%rsp
  400ea4: 48 89 e6              mov    %rsp,%rsi
  400ea7: e8 97 08 00 00        call   401743 <read_six_numbers>
  400eac: 48 89 e5              mov    %rsp,%rbp
  400eaf: 4c 8d 6c 24 0c        lea    0xc(%rsp),%r13
  400eb4: 41 bc 00 00 00 00     mov    $0x0,%r12d

  400eba: 48 89 eb              mov    %rbp,%rbx
  400ebd: 8b 45 0c              mov    0xc(%rbp),%eax
  400ec0: 39 45 00              cmp    %eax,0x0(%rbp)
  400ec3: 74 05                 je     400eca <phase_2+0x3e>
  400ec5: e8 73 07 00 00        call   40163d <explode_bomb>
  400eca: 44 03 23              add    (%rbx),%r12d
  400ecd: 48 83 c5 04           add    $0x4,%rbp
  400ed1: 4c 39 ed              cmp    %r13,%rbp
  400ed4: 75 e4                 jne    400eba <phase_2+0x2e>

  400ed6: 45 85 e4              test   %r12d,%r12d
  400ed9: 75 05                 jne    400ee0 <phase_2+0x54>
  400edb: e8 5d 07 00 00        call   40163d <explode_bomb>
  400ee0: 48 8b 5c 24 28        mov    0x28(%rsp),%rbx
  400ee5: 48 8b 6c 24 30        mov    0x30(%rsp),%rbp
  400eea: 4c 8b 64 24 38        mov    0x38(%rsp),%r12
  400eef: 4c 8b 6c 24 40        mov    0x40(%rsp),%r13
  400ef4: 48 83 c4 48           add    $0x48,%rsp
  400ef8: c3                    ret
```

通过上述分析我们可以合理猜测：程序需要我们输入 6 个数字，这 6 个数需要满足第 1 和第 4、第 2 和第 5、第 3 和第 6 个数字相等，并且前 3 个数字相加的和不能为 0。
为了印证我的猜测，阅读 *read_six_numbers*，发现额外规则：如果输入数字个数小于 6 也会触发炸弹。相应的字符串存储在 0x401eb2。

```
0000000000401743 <read_six_numbers>:
  401743: 48 83 ec 18           sub    $0x18,%rsp
  401747: 48 89 f2              mov    %rsi,%rdx
  40174a: 48 8d 4e 04           lea    0x4(%rsi),%rcx
  40174e: 48 8d 46 14           lea    0x14(%rsi),%rax
  401752: 48 89 44 24 08        mov    %rax,0x8(%rsp)
  401757: 48 8d 46 10           lea    0x10(%rsi),%rax
  40175b: 48 89 04 24           mov    %rax,(%rsp)
  40175f: 4c 8d 4e 0c           lea    0xc(%rsi),%r9
  401763: 4c 8d 46 08           lea    0x8(%rsi),%r8
  401767: be b2 1e 40 00        mov    $0x401eb2,%esi
  40176c: b8 00 00 00 00        mov    $0x0,%eax
  401771: e8 3a f3 ff ff        call   400ab0 <__isoc99_sscanf@plt>
  401776: 83 f8 05              cmp    $0x5,%eax
  401779: 7f 05                 jg     401780 <read_six_numbers+0x3d>
  40177b: e8 bd fe ff ff        call   40163d <explode_bomb>
  401780: 48 83 c4 18           add    $0x18,%rsp
  401784: c3                    ret
```

使用 GDB 调试查看内存 0x401eb2 处的内容。

```
(gdb) p (char*)0x401eb2
$2 = 0x401eb2 "%d %d %d %d %d %d"
(gdb)
```

发现是上图所示的内容，我们只需要输入满足规则的数字即可过关。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

第三关

阅读 phase_3 代码。程序首先开辟栈空间，将存储数据的地址存储到寄存器中。然后将 0x401ebe 的赋值给%esi，合理猜测这是我们需要输入的内容（后续可以使用 GDB 调试查看需要我们输入的内容是什么）。接下来是读入数据，代码保证数据有两个，否则会触发炸弹。再判断第一个输入的数据是否大于 7，大于 7 则跳转并触发炸弹，否则将第一个输入的数据搬移到寄存器%eax 中，根据公式*0x401b60+8*%eax 跳转到内存相应地址处。最后将相应的数据搬移到%eax 中，并与第二个输入的数据比较大小，如果不相等则触发炸弹，反之则过关。

```
0000000000400ef9 <phase_3>:
  400ef9: 48 83 ec 18           sub    $0x18,%rsp
  400efd: 48 8d 4c 24 08        lea    0x8(%rsp),%rcx
  400f02: 48 8d 54 24 0c        lea    0xc(%rsp),%rdx              # 开辟的栈空间位置存储到寄存器中
  400f07: be be 1e 40 00        mov    $0x401ebe,%esi              # 存储字符串的地址?
  400f0c: b8 00 00 00 00        mov    $0x0,%eax                   # 清零，累计
  400f11: e8 9a fb ff ff        call   400ab0 <__isoc99_sscanf@plt>
  400f16: 83 f8 01              cmp    $0x1,%eax                   # 读入的数据的个数保证有2个
  400f19: 7f 05                 jg     400f20 <phase_3+0x27>
  400f1b: e8 1d 07 00 00        call   40163d <explode_bomb>       # %eax <= 1则触发炸弹
  400f20: 83 7c 24 0c 07        cmpl   $0x7,0xc(%rsp)              # 地址%rsp+12的数据大于7跳转，触发炸
  400f25: 77 3c                 ja     400f63 <phase_3+0x6a>
  400f27: 8b 44 24 0c           mov    0xc(%rsp),%eax              # 地址%rsp+12的数据搬移到%eax
  400f2b: ff 24 c5 60 1b 40 00  jmp    *0x401b60(,%rax,8)          # 跳转到 (%rax*8) +0x401b60
  400f32: b8 17 02 00 00        mov    $0x217,%eax                 # 分支1
  400f37: eb 3b                 jmp    400f74 <phase_3+0x7b>
  400f39: b8 d6 00 00 00        mov    $0xd6,%eax                  # 分支2
  400f3e: eb 34                 jmp    400f74 <phase_3+0x7b>
  400f40: b8 53 01 00 00        mov    $0x153,%eax                 # 分支3
  400f45: eb 2d                 jmp    400f74 <phase_3+0x7b>
  400f47: b8 77 00 00 00        mov    $0x77,%eax                  # 分支4
  400f4c: eb 26                 jmp    400f74 <phase_3+0x7b>
  400f4e: b8 60 01 00 00        mov    $0x160,%eax                 # 分支5
  400f53: eb 1f                 jmp    400f74 <phase_3+0x7b>
  400f55: b8 97 03 00 00        mov    $0x397,%eax                 # 分支6
  400f5a: eb 18                 jmp    400f74 <phase_3+0x7b>
  400f5c: b8 9c 01 00 00        mov    $0x19c,%eax                 # 分支7
  400f61: eb 11                 jmp    400f74 <phase_3+0x7b>
  400f63: e8 d5 06 00 00        call   40163d <explode_bomb>
  400f68: b8 00 00 00 00        mov    $0x0,%eax                   # default分支?
  400f6d: eb 05                 jmp    400f74 <phase_3+0x7b>
  400f6f: b8 9e 03 00 00        mov    $0x39e,%eax                 # 分支8?
  400f74: 3b 44 24 08           cmp    0x8(%rsp),%eax              # %eax等于地址%rsp+8的数据则跳转
  400f78: 74 05                 je     400f7f <phase_3+0x86>
  400f7a: e8 be 06 00 00        call   40163d <explode_bomb>       # 不相等则触发炸弹
  400f7f: 48 83 c4 18           add    $0x18,%rsp                  # 释放申请的空间
  400f83: c3                    ret
```

启动 GDB 调试。通过输入 x 发现*（int*）（0x401b60+%rax*x）跳转的地址如下图所示。我们只需要输入的 y 和输入的 x 跳转的地址后被赋值的%eax 相等即可。该关卡一共有 8 组合法解：0 | 535, 1 | 926, 2 | 214, 3 | 339, 4 | 119, 5 | 352, 6 | 919, 7 | 412。

```
(gdb) p/x *(int*)(0x401b60 + 8 * 0)
$5 = 0x400f32
(gdb) p/x *(int*)(0x401b60 + 8 * 1)
$6 = 0x400f6f
(gdb) p/x *(int*)(0x401b60 + 8 * 2)
$7 = 0x400f39
(gdb) p/x *(int*)(0x401b60 + 8 * 3)
$8 = 0x400f40
(gdb) p/x *(int*)(0x401b60 + 8 * 4)
$9 = 0x400f47
(gdb) p/x *(int*)(0x401b60 + 8 * 5)
$10 = 0x400f4e
(gdb) p/x *(int*)(0x401b60 + 8 * 6)
$11 = 0x400f55
(gdb) p/x *(int*)(0x401b60 + 8 * 7)
$12 = 0x400f5c
(gdb)
```

使用 GDB 调试查看 0x401ebe 处的内容，发现确实是需要我们输入两个数字。

```
$12 = 0x400f5c
(gdb) p (char*)0x401ebe
$13 = 0x401ebe "%d %d"
(gdb)
```

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

第四关

阅读 phase_4 的代码。程序首先申请栈空间，将指针赋值给%rdx，将输入内容的地址存储到%esi 中，将累加器（计算输入数据的个数）清空。然后调用 scanf 函数读取数据，数据内容存放到%rdx 存储的地址中。接着判断输入数据的个数，如果不等于 1 则触发炸弹。再比较输入数据和 0 的大小，小于或等于 0 则触发炸弹，否则跳转调用函数 func4，在调用函数前将输入数据搬移到寄存器%edi 中，作为函数的参数。在函数调用结束后，比较返回值和 0x37 的大小，不相等则触发炸弹，反之过关。最后释放申请的空间。

```
0000000000400fc1 <phase_4>:
  400fc1: 48 83 ec 18           sub    $0x18,%rsp                  # 申请空间
  400fc5: 48 8d 54 24 0c        lea    0xc(%rsp),%rdx              # 将指针赋值给%rdx
  400fca: be c1 1e 40 00        mov    $0x401ec1,%esi              # 将输入内容的地址存储到%esi
  400fcf: b8 00 00 00 00        mov    $0x0,%eax                   # 清零累加器
  400fd4: e8 d7 fa ff ff        call   400ab0 <__isoc99_sscanf@plt> # 读取数据
  400fd9: 83 f8 01              cmp    $0x1,%eax                   # 比较%eax和1的大小
  400fdc: 75 07                 jne    400fe5 <phase_4+0x24>       # 不等于1则触发炸弹
  400fde: 83 7c 24 0c 00        cmpl   $0x0,0xc(%rsp)
  400fe3: 7f 05                 jg     400fea <phase_4+0x29>       # 比较输入数据和0的大小，大于0则跳转，否则触发炸弹
  400fe5: e8 53 06 00 00        call   40163d <explode_bomb>
  400fea: 8b 7c 24 0c           mov    0xc(%rsp),%edi              # 将输入数据搬移到寄存器%edi中，作为func4的参数?
  400fee: e8 91 ff ff ff        call   400f84 <func4>              # 调用函数
  400ff3: 83 f8 37              cmp    $0x37,%eax                  # 比较返回值和0x37的大小
  400ff6: 74 05                 je     400ffd <phase_4+0x3c>       # 相等则跳转
  400ff8: e8 40 06 00 00        call   40163d <explode_bomb>       # 不相等则触发炸弹
  400ffd: 48 83 c4 18           add    $0x18,%rsp                  # 释放申请的空间
  401001: c3                    ret
```

阅读 func4 的代码。首先申请空间保存即将用到的寄存器中的内容。然后将%edi 中的数据搬移到%ebx 中，将%eax 赋值为 1，作为函数无法递归到下一层时的返回值。比较参数%edi 和 1 的大小，小于或等于 1 时跳转到函数恢复寄存器内容，释放空间的阶段，否则执行两个递归函数（分别将%edi-1 和%edi-2 作为新参数）。在执行完第一个递归函数后，将返回值临时存储到%ebp 中。在执行完第二个递归函数后，将%ebp 中的数据加到%eax 中，作为当前函数的返回值。最后恢复寄存器内容并释放空间。

```
0000000000400f84 <func4>:
  400f84: 48 89 5c 24 f0        mov    %rbx,-0x10(%rsp)            # 保存寄存器中的内容
  400f89: 48 89 6c 24 f8        mov    %rbp,-0x8(%rsp)             # 保存寄存器中的内容
  400f8e: 48 83 ec 18           sub    $0x18,%rsp                  # 申请栈空间
  400f92: 89 fb                 mov    %edi,%ebx                   # 参数（phase_4中输入的数据）保存到%ebx
  400f94: b8 01 00 00 00        mov    $0x1,%eax                   # %eax=1
  400f99: 83 ff 01              cmp    $0x1,%edi                   # 比较1和参数的大小
  400f9c: 7e 14                 jle    400fb2 <func4+0x2e>         # <=1跳转，准备退出函数
  400f9e: 8d 7b ff              lea    -0x1(%rbx),%edi             # 相当于%ebi=%ebi-1
  400fa1: e8 de ff ff ff        call   400f84 <func4>
  400fa6: 89 c5                 mov    %eax,%ebp                   # 返回值搬移到%ebp中
  400fa8: 8d 7b fe              lea    -0x2(%rbx),%edi             # 相当于%edi=%edi-2
  400fab: e8 d4 ff ff ff        call   400f84 <func4>
  400fb0: 01 e8                 add    %ebp,%eax                   # 将%ebp即一个递归调用的返回值加上第二个递归调用的返回值，作为当前节点的返回值
  400fb2: 48 8b 5c 24 08        mov    0x8(%rsp),%rbx              # 恢复寄存器中的内容
  400fb7: 48 8b 6c 24 10        mov    0x10(%rsp),%rbp             # 恢复寄存器中的内容
  400fbc: 48 83 c4 18           add    $0x18,%rsp                  # 释放栈空间
  400fc0: c3                    ret
```

阅读完上述代码，我们发现 func4 是求斐波那契数列第 x 项的函数，参数代表递归层数。我们合理猜测程序需要我们输入一个数字，作为 func4 的参数，使得 fucn4（斐波那契数列）的返回值等于 0x37（换算成十进制为 55）。
综上所述，我们发现斐波那契数列的第 9 项为 55，故输入 9 即可过关。

启动 GDB 调试，查看 0x401ec1 处内容，发现确实要求输入一个整型数字。

```
$13 = 0x401ebe "%d %d"
(gdb) p (char*)0x401ec1
$14 = 0x401ec1 "%d"
(gdb)
```

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

第五关

阅读 phase_5 代码。首先移动栈指针（开辟空间），将指针赋值给寄存器，将输入内容的地址存储到寄存器，将累加器清空。然后调用 scanf 函数，判断输入数据个数是否大于 1，大于 1 则程序继续运行，否则触发炸弹。接下来将第一个数据赋值给寄存器%eax，%eax & 0xf 的操作目的是将%eax 限制在 0～15 的范围内（即第一个输入数据限制为 0～15），再判断%eax 是否为 15，如果是则触发炸弹，否则程序继续运行。其次，将%ecx、%edx 清零，准备开始循环，其中%edx 为迭代计数器，要求迭代轮数为 12 轮；%ecx 为累积器，要求每次跳转的地址处的值相加后等于第二个输入的数据。该循环是在一个表中不断跳转，初始位置即 arr[%eax]。进入循环后，将 M[%rax * 4 + 0x401ba0]的数据搬移到%eax，然后将%eax 的数据累加到%ecx 中，再判断%eax 和 0xf 是否相等，如果不相等则继续循环，否则结束循环。循环结束后，判断两个条件：迭代次数（%edx）是否等于 12 和累加数值（%ecx）是否等于第二个输入的数据，如果其中一个不满足则触发炸弹，否则过关。最后释放空间，第五关结束。

```
0000000000401002 <phase_5>:
  401002: 48 83 ec 18           sub    $0x18,%rsp                  # 移动栈指针
  401006: 48 8d 4c 24 08        lea    0x8(%rsp),%rcx              # 将指针赋值给%rcx
  40100b: 48 8d 54 24 0c        lea    0xc(%rsp),%rdx              # 将指针赋值给%rdx
  401010: be be 1e 40 00        mov    $0x401ebe,%esi              # 将输入内容的地址存储到%esi
  401015: b8 00 00 00 00        mov    $0x0,%eax                   # 清空累加器
  40101a: e8 91 fa ff ff        call   400ab0 <__isoc99_sscanf@plt> # 调用scanf函数
  40101f: 83 f8 01              cmp    $0x1,%eax                   # 比较输入数据个数和1的大小
  401022: 7f 05                 jg     401029 <phase_5+0x27>       # 大于1跳转
  401024: e8 14 06 00 00        call   40163d <explode_bomb>       # <=1触发炸弹
  401029: 8b 44 24 0c           mov    0xc(%rsp),%eax              # 将第一个输入数据搬移到%eax
  40102d: 83 e0 0f              and    $0xf,%eax                   # %eax = %eax & 0xf，判断%eax二进制低四位是否存在1
  401030: 89 44 24 0c           mov    %eax,0xc(%rsp)              # 将%eax中的数据搬移到M[%rsp+12]中（即第一个输入数据的位置）
  401034: 83 f8 0f              cmp    $0xf,%eax                   # 判断%eax-0xf是否等于0，此时0<=%eax<15
  401037: 74 2c                 je     401065 <phase_5+0x63>       # 等于0跳转，然后触发炸弹
  401039: b9 00 00 00 00        mov    $0x0,%ecx                   # %ecx=0
  40103e: ba 00 00 00 00        mov    $0x0,%edx                   # %edx=0

  401043: 83 c2 01              add    $0x1,%edx                   # %edx+=1
  401046: 48 98                 cltq                               # 拓展为8字节
  401048: 8b 04 85 a0 1b 40 00  mov    0x401ba0(,%rax,4),%eax      # 将M[%rax * 4 + 0x401ba0]的数据搬移到%eax
  40104f: 01 c1                 add    %eax,%ecx                   # %ecx = %ecx + %erax = M[%rax * 4 + 0x401ba0],合理猜测这个地方是一个数组（跳转表）
  401051: 83 f8 0f              cmp    $0xf,%eax                   # 比较%eax和0xf的大小
  401054: 75 ed                 jne    401043 <phase_5+0x41>       # 如果%eax! =0xf则跳转。循环?

  401056: 89 44 24 0c           mov    %eax,0xc(%rsp)              # 将%eax中的数据搬移到M[%rsp+12]中（即第一个输入数据的位置）
  40105a: 83 fa 0c              cmp    $0xc,%edx                   # 比较%edx和12的大小
  40105d: 75 06                 jne    401065 <phase_5+0x63>       # 不相等则触发炸弹          也即是要求循环次数为12?
  40105f: 3b 4c 24 08           cmp    0x8(%rsp),%ecx              # 比较%ecx和第二个输入数据的大小
  401063: 74 05                 je     40106a <phase_5+0x68>       # 如果等于则过关
  401065: e8 d3 05 00 00        call   40163d <explode_bomb>       # 如果不等于则触发炸弹
  40106a: 48 83 c4 18           add    $0x18,%rsp                  # 释放空间
  40106e: c3                    ret
```

进入 GDB 调试查看 0x401ba0 处～0x401ba+15 处的内容，得到下面的表格。

指针	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
数值	10	2	14	7	8	12	15	11	0	4	1	13	3	9	6	5
来源	8	10	2	12	9	15	14	3	4	13	0	7	5	11	2	6

程序需要最后一步跳到指针 6 处，一共需要跳 12 步，因此起始点应该为 7。将经过的点的数值累加起来得到 93。举例，当前指针位 6，来源为 14，说明上一次指针为 13，因为指针为 13 处的数值为 6。

进入 GDB 调试，查看 0x401ebe 处的内容，发现确实是需要我们输入两个数字。

```
$14 = 0x401ec1 "%d"
(gdb) p (char*)0x401ebe
$15 = 0x401ebe "%d %d"
(gdb) s
```

综上所述，我们只需要输入 7 和 93 即可通过关卡。

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

第六关

阅读 phase_6 代码。程序首先申请空间，对寄存器赋初始值。然后将输入的值转化为 8 字节长整型存放在内存中，返回值是一个地址，通过不断地调用之后得到对应的%rax 的值，如果最终的值与输入的%ebx 的值不同则会触发炸弹。所以需要判断 M[rax]是多少。

```
00000000004010d9 <phase_6>:
  4010d9: 48 83 ec 08           sub    $0x8,%rsp                # 申请空间
  4010dd: ba 0a 00 00 00        mov    $0xa,%edx                # %edx=0xa
  4010e2: be 00 00 00 00        mov    $0x0,%esi                # %esi=0
  4010e7: e8 94 fa ff ff        call   400b80 <strtol@plt>      # string转long
  4010ec: 89 05 8e 16 20 00     mov    %eax,0x20168e(%rip)      # 602780 <node0>
  4010f2: bf 80 27 60 00        mov    $0x602780,%edi           # 将0x602780赋值给%edi
  4010f7: e8 73 ff ff ff        call   40106f <fun6>            # 调用函数
  4010fc: 48 8b 40 08           mov    0x8(%rax),%rax
  401100: 48 8b 40 08           mov    0x8(%rax),%rax
  401104: 48 8b 40 08           mov    0x8(%rax),%rax           # %rax=M[0x8+%rax]
  401108: 8b 15 72 16 20 00     mov    0x201672(%rip),%edx      # 602780 <node0>
  40110e: 39 10                 cmp    %edx,(%rax)              # 比较M[%rax]和%edx
  401110: 74 05                 je     401117 <phase_6+0x3e>    # 相等则过关
  401112: e8 26 05 00 00        call   40163d <explode_bomb>    # 不相等则触发炸弹
  401117: 48 83 c4 08           add    $0x8,%rsp                # 释放空间
  40111b: c3                    ret
```

因为函数 func 中的代码逻辑过于晦涩难懂，所以我们使用 GDB 调试的得到我们需要的答案。首先在 0x40110e 处设置一个断点，然后输入 r 运行程序，随意输入一个数字。之后我们查看寄存器%rax 内存的指针指向的地址的内容，发现此处的值为 600。

```
(gdb) b *0x40110e
Breakpoint 1 at 0x40110e
```

```
Breakpoint 1, 0x000000000040110e in phase_6 ()
(gdb) p *($rax)
$1 = 600
```

综上所述，输入 600 即可过关。

五、实验结论：

通过解析汇编代码的作用以及不断使用 GDB 对程序进行调试之后，最终通过了所有关卡（见下图）。

```
huangliangming_2022155028@ubuntu-2204:~/Desktop$ ./bomb_64
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Science isn't about why, it's about why not?
Phase 1 defused. How about the next one?
^CSo you think you can stop the bomb with ctrl-c, do you?
^Z
[7]+  Stopped                 ./bomb_64
huangliangming_2022155028@ubuntu-2204:~/Desktop$ ./bomb_64
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Science isn't about why, it's about why not?
Phase 1 defused. How about the next one?
2 3 3 2 3 3
That's number 2.  Keep going!
3 339
Halfway there!
9
So you got that one.  Try this one.
7 93
Congratulations! You've (mostly) defused the bomb!
Hit Control-C to escape phase 6 (for free!), but if you want to
try phase 6 for extra credit, you can continue.  Just beware!
600
Congratulations! You've defused the bomb! Again!
huangliangming_2022155028@ubuntu-2204:~/Desktop$
```

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。

六、心得体会：

1. 通过本次试验，我对汇编代码有了更深刻的了解，知道如何使用汇编语言实现 if 语句，while 语句，switch 语句以及递归函数。
2. 通过本次试验，我了解了 GDB 一些常见的调试技巧该如何使用，包括 p 可以用于打印对应地址的信息，使用@+数字可以打印数组等。

指导教师批阅意见：

成绩评定：

指导教师签字：冯禹洪

2024 年　　月　　日

备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。