哈爾濱工業大學

实验报告

实 验（三）

题　　目 Binary Bomb

二进制炸弹

专　　业 计算学部

学　　号 1190200208

班　　级 1936002

学　　生 李旻翀

指 导 教 师 刘宏伟

实 验 地 点 G709

实 验 日 期 2021.4.22

计算机科学与技术学院

目 录

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；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位；

1.2.3 开发工具

GDB/OBJDUMP；EDB；KDD 等

1.3 实验预习

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。

请写出 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。

生成执行程序 sample.out。

用 gcc –S 或 CodeBlocks 或 GDB 或 OBJDUMP 等，反汇编，比较。

列出每一部分的 C 语言对应的汇编语言。

修改编译选项-O (缺省 2)、O0、O1、O2、O3，-m32/m64。再次查看生成的汇编语言与原来的区别。

注意 O1 之后无栈帧，EBP 做别的用途。-fno-omit-frame-pointer 加上栈指针。

GDB 命令详解 －tui 模式 ^XA 切换 layout 改变等等

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 hellolinux.c。反汇编查看 printf 函数的实现。

要求：C、ASM、内存(显示 hello 等内容)、堆栈（call printf 前）、寄存器同时在一个窗口。

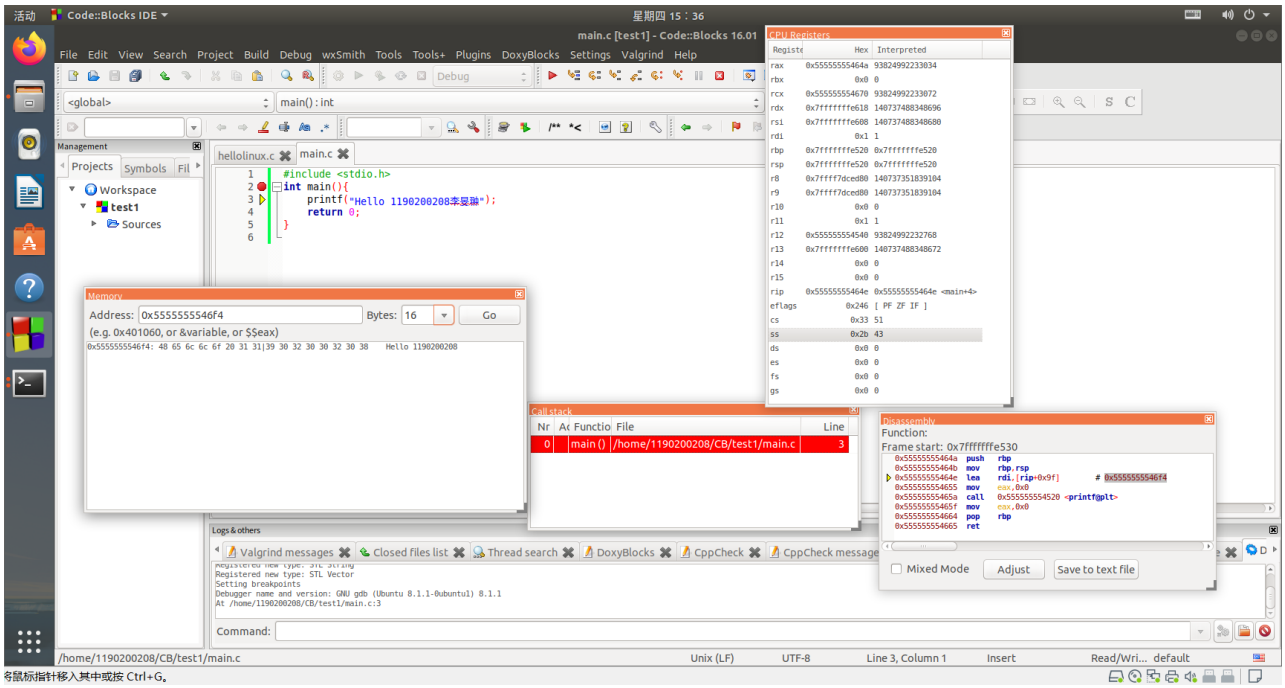


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

2.2 Ubuntu 下 EDB 运行环境建立（10 分）

用 EDB 调试 hellolinux.c 的执行文件，截图，要求同 2.1

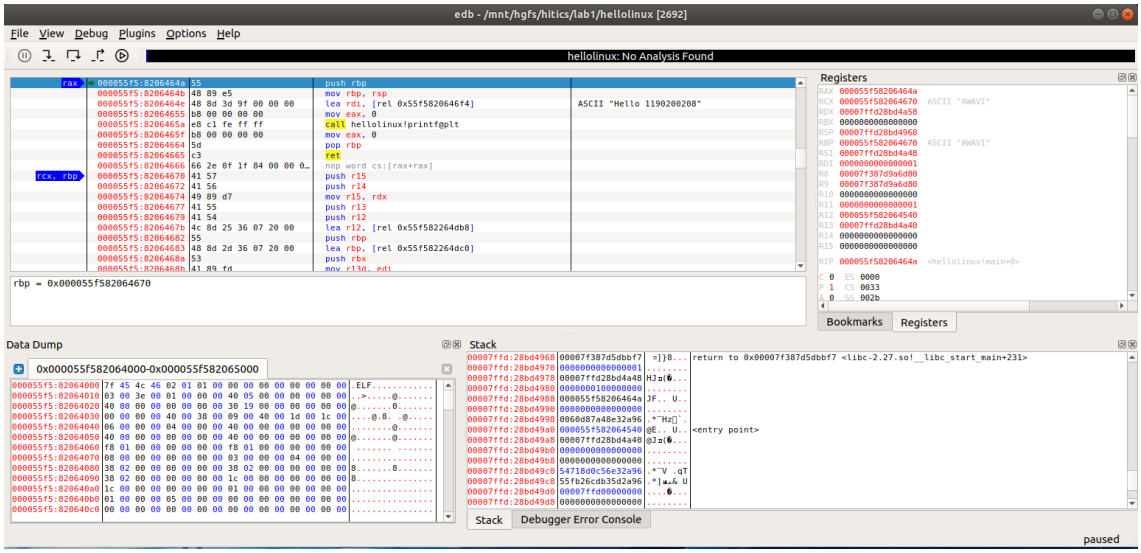


图 2-2　Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 15 分（密码 10 分，分析 5 分），总分不超过 80 分

3.1 阶段 1 的破解与分析

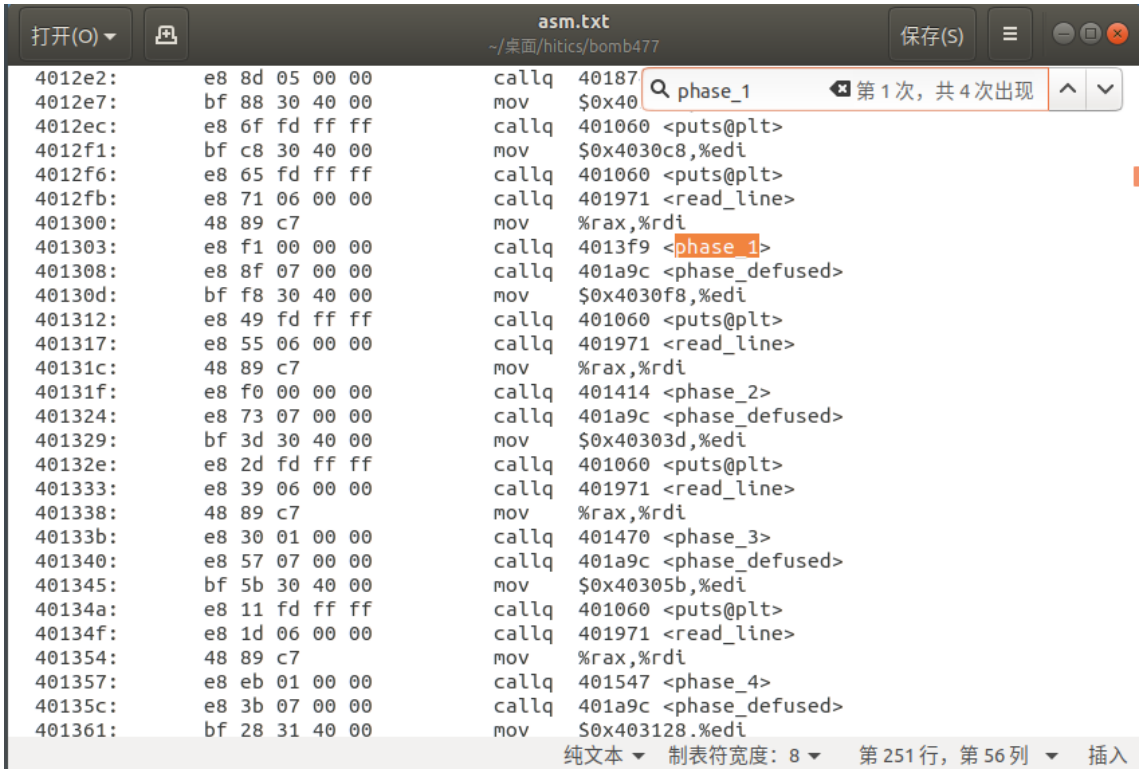
密码如下：

He is evil and fits easily into most overhead storage bins.

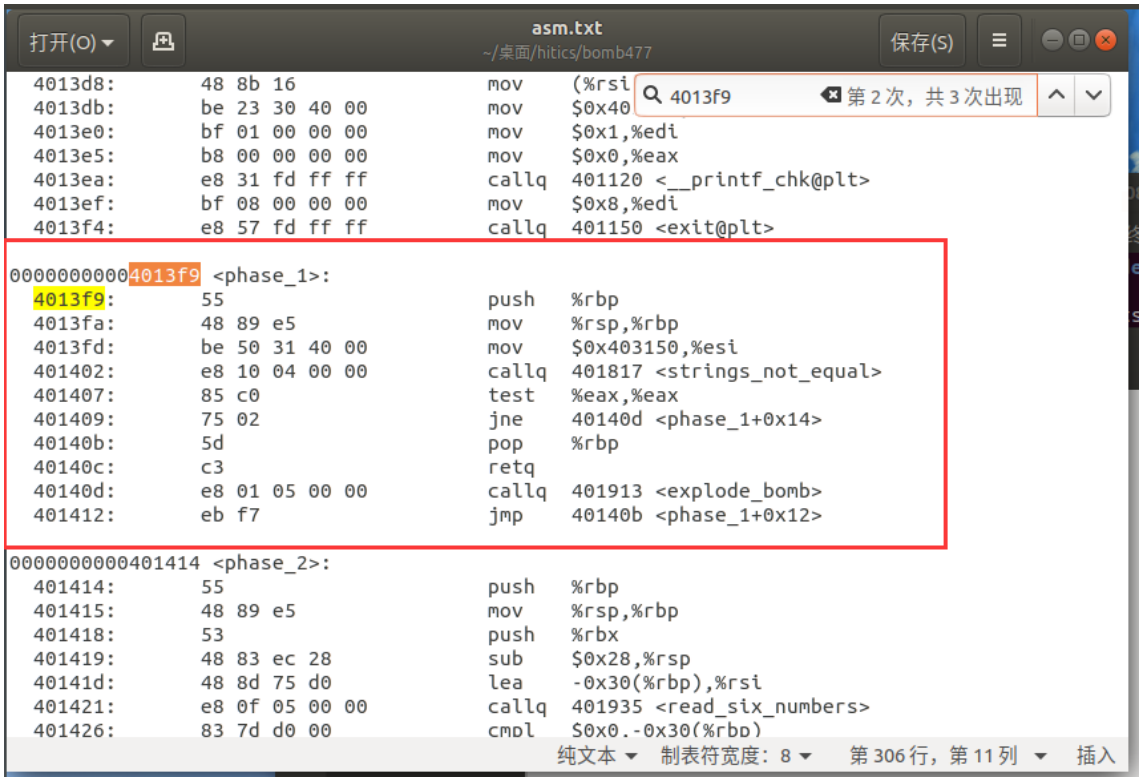
破解过程：

首先在 shell 中运行 objdump –d bomb > asm.txt，输出汇编代码

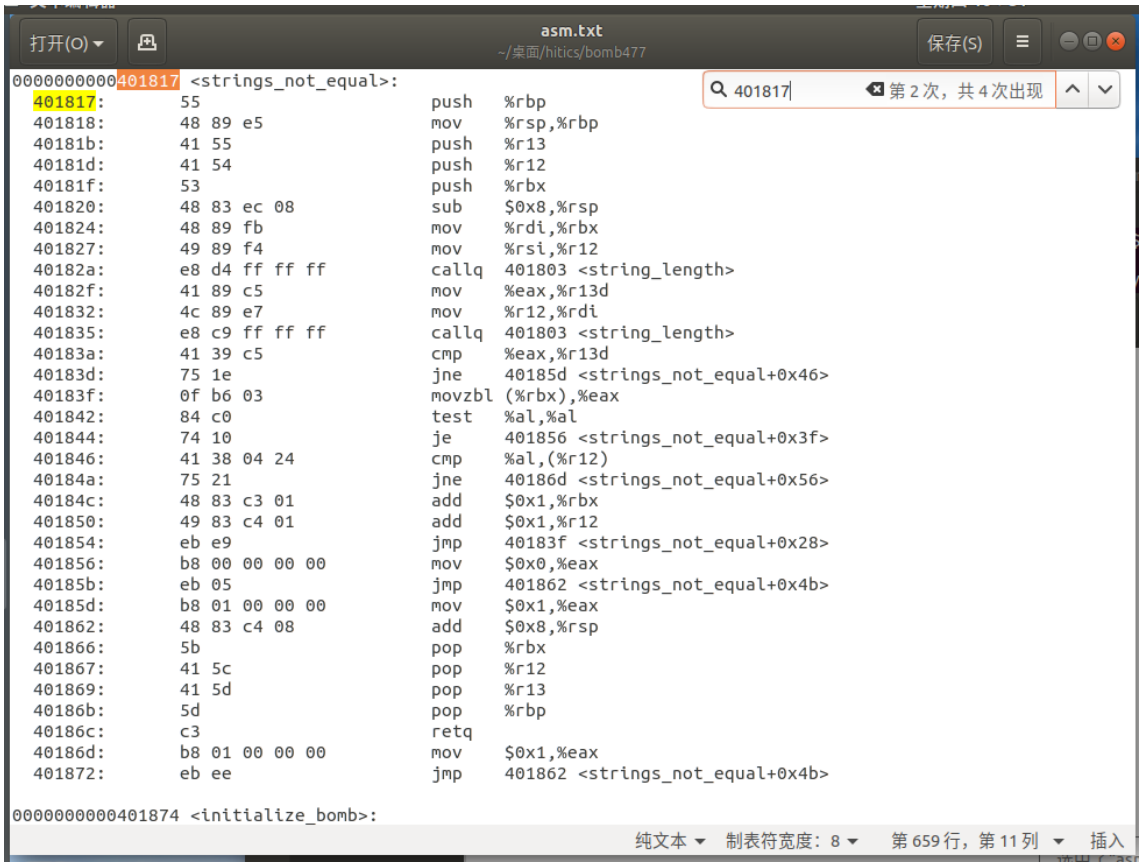
查找到 phase_1 函数的地址为 4013f9，进入查看。



phase_1 的汇编代码段：



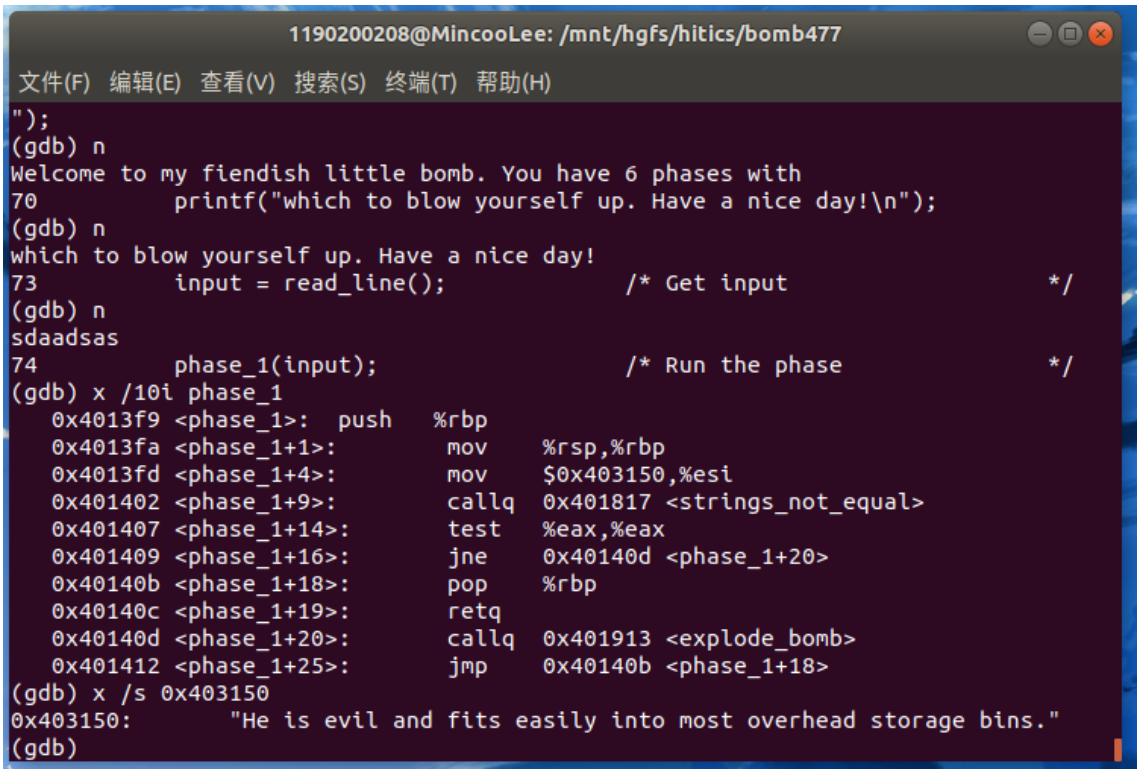
发现其中有 test 指令，根据汇编代码，若%eax 中的值不等于 0，则跳转至 explode_bomb，造成程序终止。故%eax 中的值等于 0 代表密码正确。因此进入 401817 strings_not_equal 中查看返回值：



在 main 函数中，调用了函数 read_line 读取用户输入，然后将用户输入从%rax 中转移至%rdi 中留作函数参数，而在函数 phase_1 中，将地址 0x403150 下的值传入%esi；%rdi 与%esi 中的值作为函数 strings_not_equal 的参数传入，故可以推测地址 0x403150 下保存着作为密码的字符串。

用 GDB 查看 0x403150 保存的字符串，得到密码为：

He is evil and fits easily into most overhead storage bins.



3.2 阶段 2 的破解与分析

密码如下：

0 1 1 2 3 5

破解过程：

首先查看 phase_2 对应的汇编代码。

```
0000000000401414 <phase_2>:
  401414:       55                      push   %rbp
  401415:       48 89 e5                mov    %rsp,%rbp
  401418:       53                      push   %rbx
  401419:       48 83 ec 28             sub    $0x28,%rsp
  40141d:       48 8d 75 d0             lea    -0x30(%rbp),%rsi
  401421:       e8 0f 05 00 00          callq  401935 <read_six_numbers>
  401426:       83 7d d0 00             cmpl   $0x0,-0x30(%rbp)
```

在 phase_2 引用的汇编代码中，调用了函数 read_six_numbers，由此猜测密码是六个数。进入 read_six_numbers 查看其汇编代码：

```
0000000000401935 <read_six_numbers>:
  401935:       55                      push   %rbp
  401936:       48 89 e5                mov    %rsp,%rbp
  401939:       48 89 f2                mov    %rsi,%rdx
  40193c:       48 8d 4e 04             lea    0x4(%rsi),%rcx
  401940:       48 8d 46 14             lea    0x14(%rsi),%rax
  401944:       50                      push   %rax
  401945:       48 8d 46 10             lea    0x10(%rsi),%rax
  401949:       50                      push   %rax
  40194a:       4c 8d 4e 0c             lea    0xc(%rsi),%r9
  40194e:       4c 8d 46 08             lea    0x8(%rsi),%r8
  401952:       be 43 33 40 00          mov    $0x403343,%esi
  401957:       b8 00 00 00 00          mov    $0x0,%eax
  40195c:       e8 af f7 ff ff          callq  401110 <__isoc99_sscanf@plt>
  401961:       48 83 c4 10             add    $0x10,%rsp
  401965:       83 f8 05                cmp    $0x5,%eax
  401968:       7e 02                   jle    40196c <read_six_numbers+0x37>
  40196a:       c9                      leaveq
  40196b:       c3                      retq
  40196c:       e8 a2 ff ff ff          callq  401913 <explode_bomb>
```

根据汇编代码引用了 sscanf，推测 read_six_numbers 函数的作用是读取用户输入的六个数字，根据 lea 指令的部分，判断这六个数字保存在数组中，%rax 保存数组的引用。

回到 phase_2 函数：

```
0000000000401414 <phase_2>:
  401414:       55                      push   %rbp
  401415:       48 89 e5                mov    %rsp,%rbp
  401418:       53                      push   %rbx
  401419:       48 83 ec 28             sub    $0x28,%rsp
  40141d:       48 8d 75 d0             lea    -0x30(%rbp),%rsi
  401421:       e8 0f 05 00 00          callq  401935 <read_six_numbers>
  401426:       83 7d d0 00             cmpl   $0x0,-0x30(%rbp)
  40142a:       75 06                   jne    401432 <phase_2+0x1e>
  40142c:       83 7d d4 01             cmpl   $0x1,-0x2c(%rbp)
  401430:       74 05                   je     401437 <phase_2+0x23>
  401432:       e8 dc 04 00 00          callq  401913 <explode_bomb>
  401437:       bb 02 00 00 00          mov    $0x2,%ebx
  40143c:       eb 08                   jmp    401446 <phase_2+0x32>
  40143e:       e8 d0 04 00 00          callq  401913 <explode_bomb>
  401443:       83 c3 01                add    $0x1,%ebx
  401446:       83 fb 05                cmp    $0x5,%ebx
  401449:       7f 1e                   jg     401469 <phase_2+0x55>
  40144b:       48 63 d3                movslq %ebx,%rdx
  40144e:       8d 4b fe                lea    -0x2(%rbx),%ecx
  401451:       48 63 c9                movslq %ecx,%rcx
  401454:       8d 43 ff                lea    -0x1(%rbx),%eax
  401457:       48 98                   cltq
  401459:       8b 44 85 d0             mov    -0x30(%rbp,%rax,4),%eax
  40145d:       03 44 8d d0             add    -0x30(%rbp,%rcx,4),%eax
  401461:       39 44 95 d0             cmp    %eax,-0x30(%rbp,%rdx,4)
  401465:       74 dc                   je     401443 <phase_2+0x2f>
  401467:       eb d5                   jmp    40143e <phase_2+0x2a>
  401469:       48 83 c4 28             add    $0x28,%rsp
  40146d:       5b                      pop    %rbx
  40146e:       5d                      pop    %rbp
  40146f:       c3                      retq
```

由 401426~401432 的五行关键代码，可以得知进行了两步比较操作：

首先比较 0x1 和-0x30(%rbp)，若不等，则二进制炸弹爆炸；再比较 0x1 和-0x2c(%rbp)，若相等，才会跳过函数 explode_bomb。综上所述，密码的前两个数字分别为 0 与 1。即：

$$M[-0x30 + \%rbp + 0x0] = 0;$$
$$M[-0x30 + \%rbp + 0x4] = 1;$$

之后，将%ebx 赋值为 2，并循环地将其与 5 比较，若相等则跳到最后，释放栈空间，结束循环。这也验证了我们之前密码为 6 个数字的假设。%ebx 从 2 到 5，每次循环加一。

接着阅读汇编代码，对于红框框出的这两行：

```
40144b:       48 63 d3                movslq %ebx,%rdx
40144e:       8d 4b fe                lea    -0x2(%rbx),%ecx
401451:       48 63 c9                movslq %ecx,%rcx
401454:       8d 43 ff                lea    -0x1(%rbx),%eax
```

其意义为：$$\begin{aligned} \%ecx &= \%rbx - 0x2; \\ \%eax &= \%rbx - 0x1; \end{aligned}$$

接着，由这几行，我们可以得到破解密码的关键：

```
401459:       8b 44 85 d0             mov    -0x30(%rbp,%rax,4),%eax
40145d:       03 44 8d d0             add    -0x30(%rbp,%rcx,4),%eax
401461:       39 44 95 d0             cmp    %eax,-0x30(%rbp,%rdx,4)
401465:       74 dc                   je     401443 <phase_2+0x2f>
401467:       eb d5                   jmp    40143e <phase_2+0x2a>
```

根据这几行汇编代码，可以得出两个式子：

$$\%eax = M[-0x30 + \%rbp + 4\%rax] + M[-0x30 + \%rbp + 4\%rcx];$$

$$\%eax = M[-0x30 + \%rbp + 4\%rdx];$$

进行代换，有：

$$M[-0x30 + \%rbp + 4\%rax] + M[-0x30 + \%rbp + 4\%rcx] = M[-0x30 + \%rbp + 4\%rdx];$$

依次代入$\%ebx = 2,3,4,5$，解方程，得知

$$M[-0x30 + \%rbp + 0x8] = 1;$$

$$M[-0x30 + \%rbp + 0xc] = 2;$$

$$M[-0x30 + \%rbp + 0x10] = 3;$$

$$M[-0x30 + \%rbp + 0x14] = 5;$$

再由前面所得知的：

$$M[-0x30 + \%rbp + 0x0] = 0;$$

$$M[-0x30 + \%rbp + 0x4] = 1;$$

我们可以得出，密码为 0 1 1 2 3 5，即 *Fibonacci* 数列的前 6 项。

3.3 阶段 3 的破解与分析

密码如下：存在 8 种可能的密码，分别为：

0 811

1 787

2 725

3 178

4 365

5 459

6 347

7 681

破解过程：

查看 phase_3 对应的汇编代码块：

```
0000000000401470 <phase_3>:
  401470:       55                      push   %rbp
  401471:       48 89 e5                mov    %rsp,%rbp
  401474:       48 83 ec 10             sub    $0x10,%rsp
  401478:       48 8d 4d f8             lea    -0x8(%rbp),%rcx
  40147c:       48 8d 55 fc             lea    -0x4(%rbp),%rdx
  401480:       be 4f 33 40 00          mov    $0x40334f,%esi
  401485:       b8 00 00 00 00          mov    $0x0,%eax
  40148a:       e8 81 fc ff ff          callq  401110 <__isoc99_sscanf@pl
  40148f:       83 f8 01                cmp    $0x1,%eax
  401492:       7e 11                   jle    4014a5 <phase_3+0x35>
  401494:       8b 45 fc                mov    -0x4(%rbp),%eax
  401497:       83 f8 07                cmp    $0x7,%eax
  40149a:       77 46                   ja     4014e2 <phase_3+0x72>
  40149c:       89 c0                   mov    %eax,%eax
  40149e:       ff 24 c5 c0 31 40 00    jmpq   *0x4031c0(,%rax,8)
  4014a5:       e8 69 04 00 00          callq  401913 <explode_bomb>
  4014aa:       eb e8                   jmp    401494 <phase_3+0x24>
  4014ac:       b8 2b 03 00 00          mov    $0x32b,%eax
  4014b1:       39 45 f8                cmp    %eax,-0x8(%rbp)
  4014b4:       75 3f                   jne    4014f5 <phase_3+0x85>
  4014b6:       c9                      leaveq
  4014b7:       c3                      retq
  4014b8:       b8 d5 02 00 00          mov    $0x2d5,%eax
  4014bd:       eb f2                   jmp    4014b1 <phase_3+0x41>
  4014bf:       b8 b2 00 00 00          mov    $0xb2,%eax
  4014c4:       eb eb                   jmp    4014b1 <phase_3+0x41>
  4014c6:       b8 6d 01 00 00          mov    $0x16d,%eax
  4014cb:       eb e4                   jmp    4014b1 <phase_3+0x41>
  4014cd:       b8 cb 01 00 00          mov    $0x1cb,%eax
  4014d2:       eb dd                   jmp    4014b1 <phase_3+0x41>
  4014d4:       b8 5b 01 00 00          mov    $0x15b,%eax
  4014d9:       eb d6                   jmp    4014b1 <phase_3+0x41>
  4014db:       b8 a9 02 00 00          mov    $0x2a9,%eax
  4014e0:       eb cf                   jmp    4014b1 <phase_3+0x41>
  4014e2:       e8 2c 04 00 00          callq  401913 <explode_bomb>
  4014e7:       b8 00 00 00 00          mov    $0x0,%eax
  4014ec:       eb c3                   jmp    4014b1 <phase_3+0x41>
  4014ee:       b8 13 03 00 00          mov    $0x313,%eax
  4014f3:       eb bc                   jmp    4014b1 <phase_3+0x41>
  4014f5:       e8 19 04 00 00          callq  401913 <explode_bomb>
  4014fa:       eb ba                   jmp    4014b6 <phase_3+0x46>
```

由 401478 ~ 401480 三行代码，猜测密码由两个值组成，分别保存在 $M[\%rbp\ -0x4]$ 与 $M[\%rbp\ -0x8]$ 中。

```
401478:       48 8d 4d f8             lea    -0x8(%rbp),%rcx
40147c:       48 8d 55 fc             lea    -0x4(%rbp),%rdx
401480:       be 4f 33 40 00          mov    $0x40334f,%esi
```

打开 GDB，查看 0x40334f 地址的内容如下，确定密码为两个整形数。

```
(gdb) x /s 0x40334f
0x40334f:       "%d %d"
```

由 401494 ~ 40149a 三行代码可知，$M[\%rbp\ -0x4]$ 所保存的密码是小于等于 7 的无符号数，即 0,1,2,3,4,5,6,7。

```
401494:       8b 45 fc                mov    -0x4(%rbp),%eax
401497:       83 f8 07                cmp    $0x7,%eax
40149a:       77 46                   ja     4014e2 <phase_3+0x72>
```

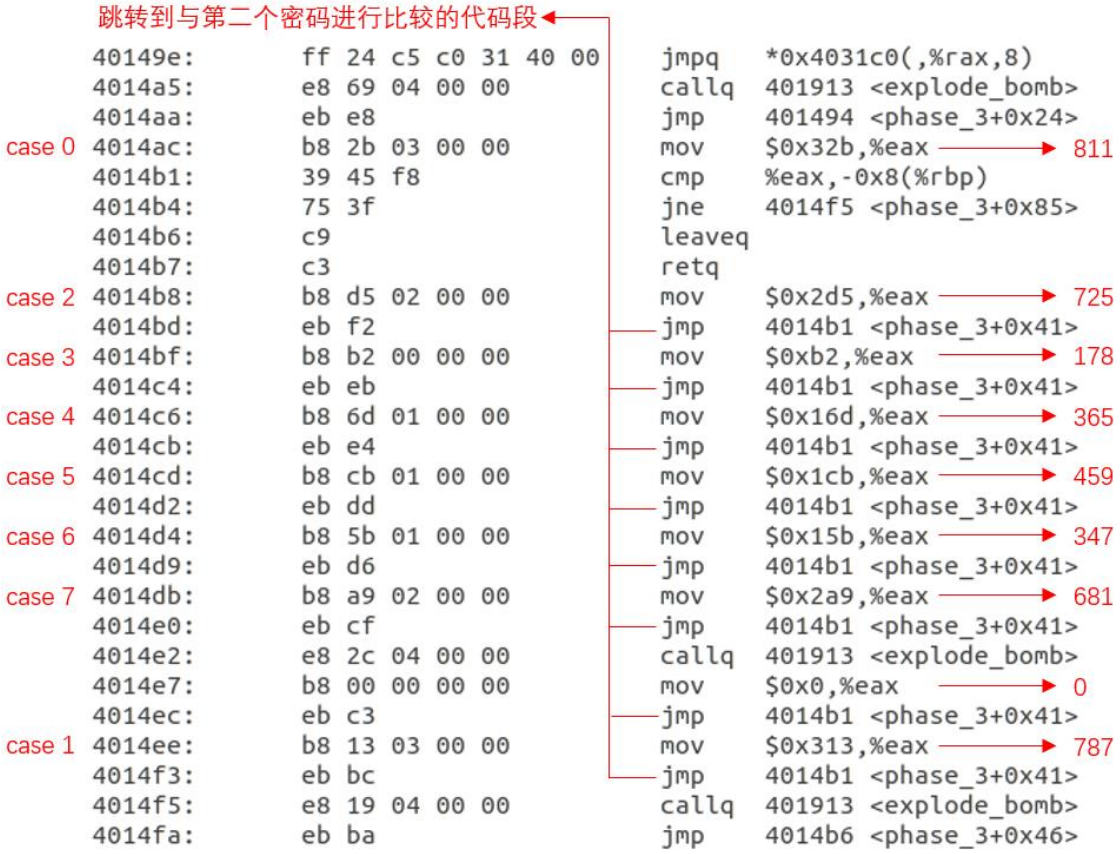
由 40149e 行可以知道，汇编代码段用到了 switch 语句，0x4031c0 为跳转表的位置。

```
40149e:       ff 24 c5 c0 31 40 00    jmpq   *0x4031c0(,%rax,8)
```

通过 GDB 查看跳转表内容，得到如图所示结果：

```
(gdb) x /16wx 0x4031c0
0x4031c0:       0x004014ac      0x00000000      0x004014ee      0x00000000
0x4031d0:       0x004014b8      0x00000000      0x004014bf      0x00000000
0x4031e0:       0x004014c6      0x00000000      0x004014cd      0x00000000
0x4031f0:       0x004014d4      0x00000000      0x004014db      0x00000000
```

因此，switch 的跳转结构如下：



综上，存在 8 种可能的密码，分别为：

0 811

1 787

2 725

3 178

4 365

5 459

6 347

7 681

3.4 阶段 4 的破解与分析

密码如下：

密码有三组：

108 2

162 3

216 4

破解过程：

查看 phase_4 对应汇编代码段：

```
0000000000401547 <phase_4>:
  401547:       55                      push   %rbp
  401548:       48 89 e5                mov    %rsp,%rbp
  40154b:       48 83 ec 10             sub    $0x10,%rsp
  40154f:       48 8d 4d fc             lea    -0x4(%rbp),%rcx  P2
  401553:       48 8d 55 f8             lea    -0x8(%rbp),%rdx  P1
  401557:       be 4f 33 40 00          mov    $0x40334f,%esi
  40155c:       b8 00 00 00 00          mov    $0x0,%eax
  401561:       e8 aa fb ff ff          callq  401110 <__isoc99_sscanf@plt>
  401566:       83 f8 02                cmp    $0x2,%eax
  401569:       75 0d                   jne    401578 <phase_4+0x31>
  40156b:       8b 45 fc                mov    -0x4(%rbp),%eax
  40156e:       83 f8 01                cmp    $0x1,%eax
  401571:       7e 05                   jle    401578 <phase_4+0x31>
  401573:       83 f8 04                cmp    $0x4,%eax
  401576:       7e 05                   jle    40157d <phase_4+0x36>
  401578:       e8 96 03 00 00          callq  401913 <explode_bomb>
  40157d:       8b 75 fc                mov    -0x4(%rbp),%esi
  401580:       bf 08 00 00 00          mov    $0x8,%edi
  401585:       e8 72 ff ff ff          callq  4014fc <func4>
  40158a:       39 45 f8                cmp    %eax,-0x8(%rbp)
  40158d:       75 02                   jne    401591 <phase_4+0x4a>
  40158f:       c9                      leaveq
  401590:       c3                      retq
  401591:       e8 7d 03 00 00          callq  401913 <explode_bomb>
  401596:       eb f7                   jmp    40158f <phase_4+0x48>
```

打开 GDB，查看 0x40334f 地址的内容如下，确定密码为两个整形数，分别记为$P1,P2$。

```
(gdb) x /s 0x40334f
0x40334f:       "%d %d"
```

对于这四行代码，在调用__isoc99_sscanf@plt 后，%eax 中存储着输入数值的个数，若输入不是两个数，则直接跳转到炸弹。

```
40155c:       b8 00 00 00 00          mov    $0x0,%eax
401561:       e8 aa fb ff ff          callq  401110 <__isoc99_sscanf@plt>
401566:       83 f8 02                cmp    $0x2,%eax
401569:       75 0d                   jne    401578 <phase_4+0x31>
```

由 40156b ~ 401578 可知，$P2$ 必须是大于 1，小于等于 4 的整数，即 2,3,4 中

的一个。

```
40156b:       8b 45 fc                mov    -0x4(%rbp),%eax
40156e:       83 f8 01                cmp    $0x1,%eax
401571:       7e 05                   jle    401578 <phase_4+0x31>
401573:       83 f8 04                cmp    $0x4,%eax
401576:       7e 05                   jle    40157d <phase_4+0x36>
401578:       e8 96 03 00 00          callq  401913 <explode_bomb>
```

%edi 与%esi 分别是传入$func4$的两个参数， %edi 值为 0x8，%esi 值为$P2$。记两个参数分别为$C1,C2$。有$C1=0x8,C2=P2$。

```
40157d:       8b 75 fc                mov    -0x4(%rbp),%esi
401580:       bf 08 00 00 00          mov    $0x8,%edi
401585:       e8 72 ff ff ff          callq  4014fc <func4>
```

由 401585 ~ 401590 可知， $func4$的返回值即为$P1$。

```
401585:       e8 72 ff ff ff          callq  4014fc <func4>
40158a:       39 45 f8                cmp    %eax,-0x8(%rbp)
40158d:       75 02                   jne    401591 <phase_4+0x4a>
40158f:       c9                      leaveq
401590:       c3                      retq
```

进入$func4$函数查看汇编代码：

```
00000000004014fc <func4>:
  4014fc:       85 ff                   test   %edi,%edi
  4014fe:       7e 3d                   jle    40153d <func4+0x41>
  401500:       55                      push   %rbp
  401501:       48 89 e5                mov    %rsp,%rbp
  401504:       41 55                   push   %r13
  401506:       41 54                   push   %r12
  401508:       53                      push   %rbx
  401509:       48 83 ec 08             sub    $0x8,%rsp
  40150d:       41 89 fc                mov    %edi,%r12d
  401510:       89 f3                   mov    %esi,%ebx
  401512:       83 ff 01                cmp    $0x1,%edi
  401515:       74 2c                   je     401543 <func4+0x47>
  401517:       8d 7f ff                lea    -0x1(%rdi),%edi
  40151a:       e8 dd ff ff ff          callq  4014fc <func4>
  40151f:       44 8d 2c 18             lea    (%rax,%rbx,1),%r13d
  401523:       41 8d 7c 24 fe          lea    -0x2(%r12),%edi
  401528:       89 de                   mov    %ebx,%esi
  40152a:       e8 cd ff ff ff          callq  4014fc <func4>
  40152f:       44 01 e8                add    %r13d,%eax
  401532:       48 83 c4 08             add    $0x8,%rsp
  401536:       5b                      pop    %rbx
  401537:       41 5c                   pop    %r12
  401539:       41 5d                   pop    %r13
  40153b:       5d                      pop    %rbp
  40153c:       c3                      retq
  40153d:       b8 00 00 00 00          mov    $0x0,%eax
  401542:       c3                      retq
  401543:       89 f0                   mov    %esi,%eax
  401545:       eb eb                   jmp    401532 <func4+0x36>
```

若C1为0，则return 0

若C1为1，则return C2

再次调用func4,
%eax = func4 (C1-1,C2)

再次调用func4,
%eax = func4 (C1-2,C2)

最终的返回值
%eax = func4 (C1-2,C2)+func4(C1-1,C2)+C2

进行代换，我们可以得到如下公式：

$$P1 = func4(8,P2) = func4(6,P2) + func4(7,P2) + P2 \qquad P2 = 2,3,4$$

代入$P2 = 2,3,4$，进行迭代，可以得到密码有三组：

108 2

162 3

216 4

3.5 阶段 5 的破解与分析

密码如下：

5 115 （其中的 5 可以替换成 16 进制下末位为 5 的任意数，如 21(0x15)等）

破解过程：

查看 phase_5 的汇编代码：

```
0000000000401598 <phase_5>:
  401598:       55                      push   %rbp
  401599:       48 89 e5                mov    %rsp,%rbp
  40159c:       48 83 ec 10             sub    $0x10,%rsp
  4015a0:       48 8d 4d f8             lea    -0x8(%rbp),%rcx  P2
  4015a4:       48 8d 55 fc             lea    -0x4(%rbp),%rdx  P1
  4015a8:       be 4f 33 40 00          mov    $0x40334f,%esi
  4015ad:       b8 00 00 00 00          mov    $0x0,%eax
  4015b2:       e8 59 fb ff ff          callq  401110 <__isoc99_sscanf@plt>
  4015b7:       83 f8 01                cmp    $0x1,%eax
  4015ba:       7e 2e                   jle    4015ea <phase_5+0x52>
  4015bc:       8b 45 fc                mov    -0x4(%rbp),%eax
  4015bf:       83 e0 0f                and    $0xf,%eax
  4015c2:       89 45 fc                mov    %eax,-0x4(%rbp)
  4015c5:       b9 00 00 00 00          mov    $0x0,%ecx
  4015ca:       ba 00 00 00 00          mov    $0x0,%edx
  4015cf:       8b 45 fc                mov    -0x4(%rbp),%eax
  4015d2:       83 f8 0f                cmp    $0xf,%eax
  4015d5:       74 1a                   je     4015f1 <phase_5+0x59>
  4015d7:       83 c2 01                add    $0x1,%edx
  4015da:       48 98                   cltq
  4015dc:       8b 04 85 00 32 40 00    mov    0x403200(,%rax,4),%eax
  4015e3:       89 45 fc                mov    %eax,-0x4(%rbp)
  4015e6:       01 c1                   add    %eax,%ecx
  4015e8:       eb e5                   jmp    4015cf <phase_5+0x37>
  4015ea:       e8 24 03 00 00          callq  401913 <explode_bomb>
  4015ef:       eb cb                   jmp    4015bc <phase_5+0x24>
  4015f1:       83 fa 0f                cmp    $0xf,%edx
  4015f4:       75 05                   jne    4015fb <phase_5+0x63>
  4015f6:       39 4d f8                cmp    %ecx,-0x8(%rbp)
  4015f9:       74 05                   je     401600 <phase_5+0x68>
  4015fb:       e8 13 03 00 00          callq  401913 <explode_bomb>
  401600:       c9                      leaveq
  401601:       c3                      retq
```

打开 GDB，查看 0x40334f 地址的内容如下，确定密码为两个整形数，分别记为$P1$,$P2$。

```
(gdb) x /s 0x40334f
0x40334f:       "%d %d"
```

4015bf ~ 4015c2 行对应的代码表示$M[\%rbp-0x4]$保存十六进制数$P1$的最后一位。

```
4015bf:       83 e0 0f                and    $0xf,%eax
4015c2:       89 45 fc                mov    %eax,-0x4(%rbp)
```

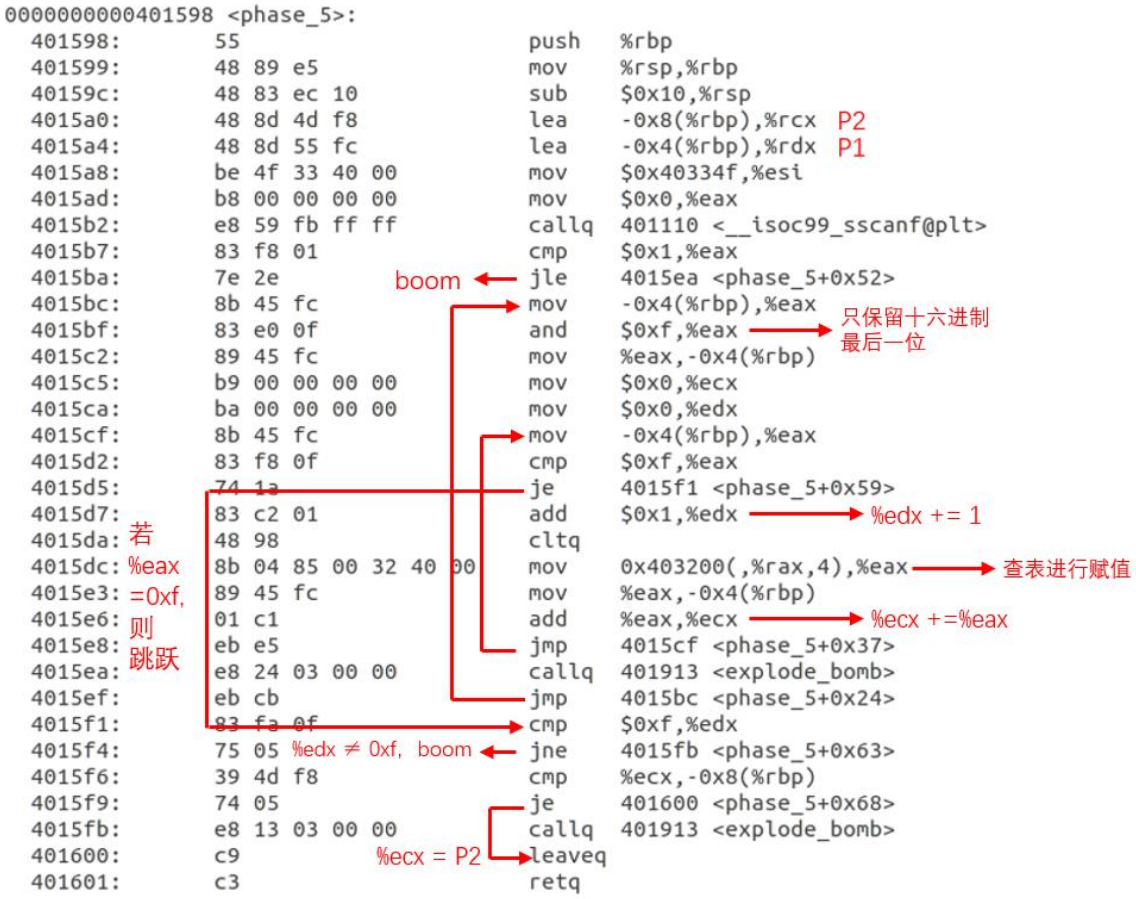
由 4015bc ~ 4015d5 可知，$P1$的最后一位不为 f，否则会引爆炸弹。

```
4015bc:       8b 45 fc                mov    -0x4(%rbp),%eax
4015bf:       83 e0 0f                and    $0xf,%eax
4015c2:       89 45 fc                mov    %eax,-0x4(%rbp)
4015c5:       b9 00 00 00 00          mov    $0x0,%ecx
4015ca:       ba 00 00 00 00          mov    $0x0,%edx
4015cf:       8b 45 fc                mov    -0x4(%rbp),%eax
4015d2:       83 f8 0f                cmp    $0xf,%eax
4015d5:       74 1a                   je     4015f1 <phase_5+0x59>
```

查看 0x403200 对应的内容，发现从 0x403200 开始保存着 16 个值。值通过$M[0x403200+4*rax]$的方式访问。

```
(gdb) x /16wx 0x403200
0x403200 <array.3401>:  0x0000000a      0x00000002      0x0000000e      0x00000007
0x403210 <array.3401+16>:       0x00000008      0x0000000c      0x0000000f      0x0000000b
0x403220 <array.3401+32>:       0x00000000      0x00000004      0x00000001      0x0000000d
0x403230 <array.3401+48>:       0x00000003      0x00000009      0x00000006      0x00000005
(gdb) x /20wx 0x403200
0x403200 <array.3401>:  0x0000000a      0x00000002      0x0000000e      0x00000007
0x403210 <array.3401+16>:       0x00000008      0x0000000c      0x0000000f      0x0000000b
0x403220 <array.3401+32>:       0x00000000      0x00000004      0x00000001      0x0000000d
0x403230 <array.3401+48>:       0x00000003      0x00000009      0x00000006      0x00000005
0x403240:       0x79206f53      0x7420756f      0x6b6e6968      0x756f7920
```

对 phase_5 程序进行整体分析：



可知，只有当同时满足条件：$\%eax = 0xf$，$\%edx = 0xf$时，程序才能正常退出，此时$\%ecx = P2$。$\%ecx$保存着每次查表赋值后除了第一个值之外所有值的总和，而$\%edx$为计数变量，从 0 开始，每执行一次查表赋值的过程便自加一，由此可见，查表赋值的过程共执行了 16 次。

表中的数值转换有如下规律：

$0 \to a$　　$8 \to 0$

$1 \to 2$　　$9 \to 4$

$2 \to e$　　$a \to 1$

$3 \to 7$　　$b \to d$

$4 \to 8$　　$c \to 3$

$5 \to c$　　$d \to 9$

$6 \to f$　　$e \to 6$

$7 \to b$　　$f \to 5$

要进行 16 次循环，且最后得到$\%eax = 0xf$，则变化情况只可能是：

$5 \to c \to 3 \to 7 \to b \to d \to 9 \to 4 \to 8 \to 0 \to a \to 1 \to 2 \to e \to 6 \to f$

需要注意的是，在第一次执行过程中，$\%ecx$不增加，即意味着在第一次执行时，$\%ecx$不会加上 5。

所以可以得到：

$P1 = 5$（或者16进制下末位为5的任意数）

$$P2 = \sum_{i=0}^{f} i - 0x5 = 0x73 = 115_{(10)}$$

3.6 阶段 6 的破解与分析

密码如下：

破解过程：

由于 phase_6 的汇编代码过长，将其分为几个模块来分析：

第一部分，初始化

在这一部分中国，完成了用户输入的读入，开辟栈空间，并初始化$\%r12d = 0$，根据其对 read_six_numbers 函数的调用可以确定 phase_6 密码为 6 个数。该部分只执行一次，便 jump 到之后的代码段。

```
0000000000401602 <phase_6>:
  401602:       55                      push   %rbp
  401603:       48 89 e5                mov    %rsp,%rbp
  401606:       41 55                   push   %r13
  401608:       41 54                   push   %r12
  40160a:       53                      push   %rbx
  40160b:       48 83 ec 58             sub    $0x58,%rsp
  40160f:       48 8d 75 c0             lea    -0x40(%rbp),%rsi
  401613:       e8 1d 03 00 00          callq  401935 <read_six_numbers>
  401618:       41 bc 00 00 00 00       mov    $0x0,%r12d
  40161e:       eb 29                   jmp    401649 <phase_6+0x47>
```

第二部分，初步确定密码的可能情况

根据 40161e 行的代码，程序在执行完第一部分初始化后跳转到 401649 行，开始进行如下操作：

```
401649:       41 83 fc 05             cmp    $0x5,%r12d
40164d:       7f 19                   jg     401668 <phase_6+0x66>
40164f:       49 63 c4                movslq %r12d,%rax
401652:       8b 44 85 c0             mov    -0x40(%rbp,%rax,4),%eax
401656:       83 e8 01                sub    $0x1,%eax
401659:       83 f8 05                cmp    $0x5,%eax
40165c:       77 c2          boom!    ja     401620 <phase_6+0x1e>
40165e:       45 8d 6c 24 01          lea    0x1(%r12),%r13d
401663:       44 89 eb                mov    %r13d,%ebx
401666:       eb c2                   jmp    40162a <phase_6+0x28>
```

初次执行，该代码段无用

初次赋值 $\%eax = M[\%rbp - 0x40]$

$\%eax --= 1$

$\%r13d = \%r12 + 0x1 = 0x1$

$\%ebx = 0x1$

根据 40165c 行可知，$M[\%rbp - 0x40]$ 处的值为小于等于 6 的数。

在跳转到 40162a 行前，各寄存器的值如下：

$$\%eax = M[\%rbp - 0x40] - 1$$
$$\%r12d = 0x0$$
$$\%r13d = 0x1$$
$$\%ebx = 0x1$$

跳转到 40162a 行，执行如下操作：

```
401627:       83 c3 01                add    $0x1,%ebx
40162a:       83 fb 05                cmp    $0x5,%ebx
40162d:       7f 17                   jg     401646 <phase_6+0x44>
40162f:       49 63 c4                movslq %r12d,%rax
401632:       48 63 d3                movslq %ebx,%rdx
401635:       8b 7c 95 c0    跳出     mov    -0x40(%rbp,%rdx,4),%edi
401639:       39 7c 85 c0             cmp    %edi,-0x40(%rbp,%rax,4)
40163d:       75 e8                   jne    401627 <phase_6+0x25>
40163f:       e8 cf 02 00 00          callq  401913 <explode_bomb>
```

$\%ebx += 0x1$，初始$\%ebx = 0x1$

$\%ebx = 0x6$时跳出，说明循环一共执行5次，正好对应着剩下的5个密码

每次比较$M[-0x40 + \%rbp + 4\%rdx]$与$M[-0x40 + \%rbp + 4\%rax]$

相当于比较$M[-0x40 + \%rbp + 4\%ebx]$与$M[-0x40 + \%rbp + 0]$

说明剩下五个密码都与第一个密码不同

在跳转到 401646 行前，主要的寄存器值如下（后续会用到）：

$$\%r13d = 0x1$$

跳转到 401646 行时，再次执行如下部分，执行完毕后，又跳转到 40162a 行，于是程序在 401646,401642 两个代码块间反复跳跃。$\%r13d, \%r12d, \%ebx$ 作为计数变量，控制着这个过程的执行次数。

```
401646:       45 89 ec                mov    %r13d,%r12d
401649:       41 83 fc 05             cmp    $0x5,%r12d
40164d:       7f 19                   jg     401668 <phase_6+0x66>
40164f:       49 63 c4                movslq %r12d,%rax
401652:       8b 44 85 c0             mov    -0x40(%rbp,%rax,4),%eax
401656:       83 e8 01                sub    $0x1,%eax
401659:       83 f8 05                cmp    $0x5,%eax
40165c:       77 c2          boom!    ja     401620 <phase_6+0x1e>
40165e:       45 8d 6c 24 01          lea    0x1(%r12),%r13d
401663:       44 89 eb                mov    %r13d,%ebx
401666:       eb c2                   jmp    40162a <phase_6+0x28>
```

$\%r12d = 0x1$

$\%rax = 0x1$

$\%eax = M[-0x40 + \%rbp + 0x4]$

$\%eax --= 1$

$\%r13d = \%r12 + 0x1 = 0x2$

$\%ebx = 0x2$

分析可知，上述两部分反复执行的代码块说明六个密码均是小于等于 6 的非零正数，且互不相同。也就是说，密码是$1, 2, 3, 4, 5, 6$的某种排列。

直到上述部分的最后一次循环结束，程序进入 401668 行及以后的代码块，不会再跳转回前面的部分，程序开始执行新一阶段的内容。

第三部分，确定密码

从 401668 开始，通过跳转关系确定一段表示循环的汇编代码，分析该段循环表示的意义：

```
401668:   be 00 00 00 00      mov    $0x0,%esi
40166d:   eb 08               jmp    401677 <phase_6+0x75>
40166f:   48 89 54 cd 90      mov    %rdx,-0x70(%rbp,%rcx,8)
401674:   83 c6 01            add    $0x1,%esi
401677:   83 fe 05            cmp    $0x5,%esi
40167a:   7f 1c               jg     401698 <phase_6+0x96>  要想跳出该层循环，必须满足%esi＞0x5，即%esi＝0x6
40167c:   b8 01 00 00 00      mov    $0x1,%eax  %eax＝0x1
401681:   ba d0 52 40 00      mov    $0x4052d0,%edx  %edx＝0x4052d0，为一个地址，其内容见下图
401686:   48 63 ce            movslq %esi,%rcx
401689:   39 44 8d c0         cmp    %eax,-0x40(%rbp,%rcx,4)  %eax与M[-0x40＋%rbp＋4%rcx]比较
40168d:   7e e0               jle    40166f <phase_6+0x6d>    若%eax≥M[-0x40＋%rbp＋4%rcx]，则跳转
40168f:   48 8b 52 08         mov    0x8(%rdx),%rdx  %rdx＋＝0x8，地址后移0x8，读取下一段内容
401693:   83 c0 01            add    $0x1,%eax  %eax＋＝0x1
401696:   eb ee               jmp    401686 <phase_6+0x84>
401698:   48 8b 5d 90         mov    -0x70(%rbp),%rbx
```

初次进入该段循环时，运行情况分析如下：

置$\%esi=0x0$,$\%eax=0x1$，然后令$\%edx$保存地址 0x4052d0，进入 GDB 查看该地址内容如下：

```
(gdb) x /24xw 0x4052d0
0x4052d0 <node1>:    0x00000087    0x00000001    0x004052e0    0x00000000
0x4052e0 <node2>:    0x00000100    0x00000002    0x004052f0    0x00000000
0x4052f0 <node3>:    0x000001c2    0x00000003    0x00405300    0x00000000
0x405300 <node4>:    0x0000003a    0x00000004    0x00405310    0x00000000
0x405310 <node5>:    0x000001ea    0x00000005    0x00405320    0x00000000
0x405320 <node6>:    0x00000329    0x00000006    0x00000000    0x00000000
```

可见 0x4052d0 依次保存着 6 个 node。初始情况下，$\%edx$保存的地址对应着数 0x87。

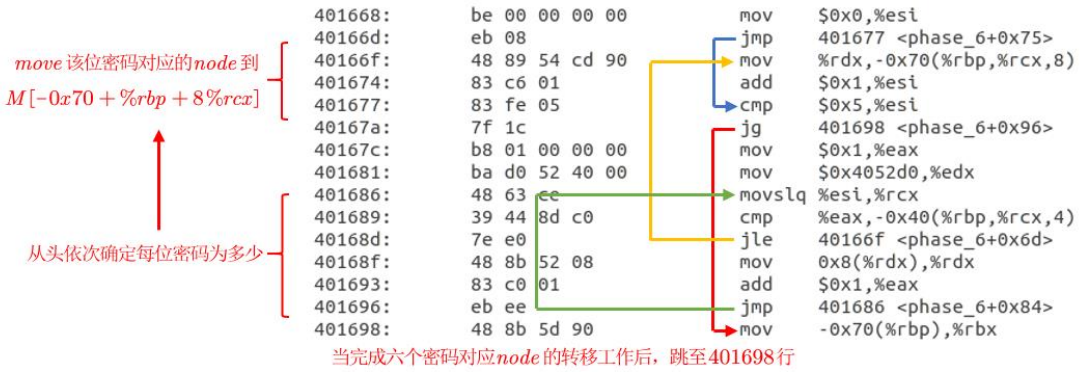
对于第一次的循环，401689 与 40168d 两行的意义是比较$M[-0x40+\%rbp]$与$\%eax$，若$\%eax\geq M[-0x40+\%rbp]$，即第一个密码小于等于 1，则跳至 40166f 行，否则程序继续按顺序运行，使$\%edx$的地址后移 0x8，指向 0x100，$\%eax$自加一。

在 40168d 行未发生跳转时，第二次循环的执行情况如下：

再次比较$M[-0x40+\%rbp]$与$\%eax$，若第一个密码小于等于 2，则跳至 40166f 行，否则程序继续按顺序运行，使$\%edx$的地址再后移 0x8，指向 0x1c2，$\%eax$

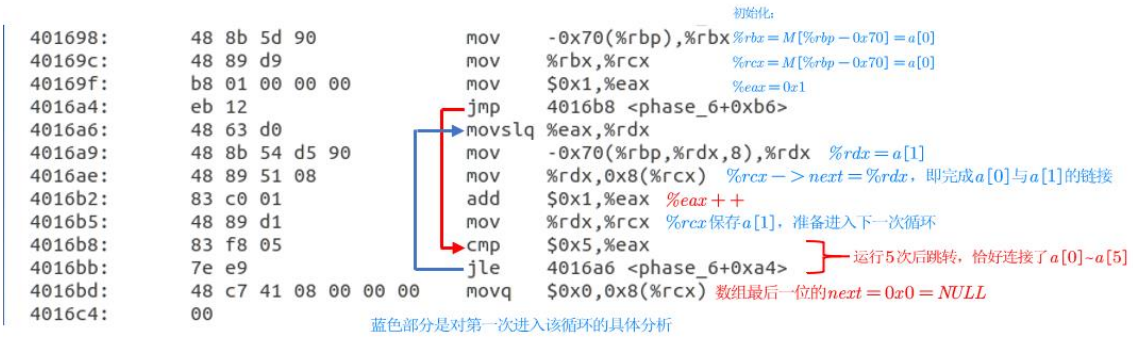
自加一。

结合 40166f 行汇编代码，可以得出如下结论：

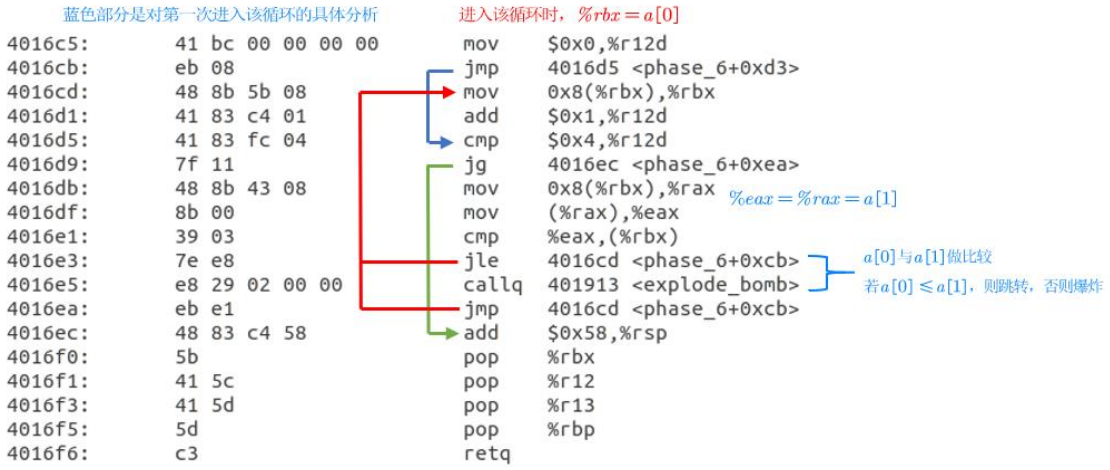


可知该段代码的作用：对于第 i 个密码，将其对应的第 i 个 node 转移至 $M[-0x70+\%rbp+8i]$。

之后进入下一阶段。该阶段作用相当于将 node 放入一个六位的数组$a[6]$，并从头到尾将其连接起来。



进入最后一个阶段，对于第一次进入该循环，分析如下：



在完成第一次循环后，程序跳至 4016cd 行，$\%rbx$ 指向 $a[1]$，开始下一次比较。分析可知，密码应满足的条件是：后一个密码对应的 node 值大于等于前一个密码对应的 node 值。

```
(gdb) x /24xw 0x4052d0        大小关系（最小为1）
0x4052d0 <node1>:       0x00000087  2       0x00000001      0x004052e0      0x00000000
0x4052e0 <node2>:       0x00000100  3       0x00000002      0x004052f0      0x00000000
0x4052f0 <node3>:       0x000001c2  4       0x00000003      0x00405300      0x00000000
0x405300 <node4>:       0x0000003a  1       0x00000004      0x00405310      0x00000000
0x405310 <node5>:       0x000001ea  5       0x00000005      0x00405320      0x00000000
0x405320 <node6>:       0x00000329  6       0x00000006      0x00000000      0x00000000
```

根据 GDB 查看 node 值，又由于每一位密码互不相同且密码是 $1,2,3,4,5,6$ 的某种排列，可知密码为 4 1 2 3 5 6。

```
1190200208@MincooLee:~/桌面/hitics/bomb477$ ./bomb ans.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...
Congratulations! You've defused the bomb!
1190200208@MincooLee:~/桌面/hitics/bomb477$
```

验证无误，六个炸弹破解成功。

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. 对汇编代码的作用，原理有了更为深刻的认识，阅读汇编代码的能力有了极大提高。
2. 本次实验“寓教于乐”的方式非常棒，让我对计算机系统的后续实验产生了更大的兴趣。

4.2 请给出对本次实验内容的建议

实验很好很经典，游戏体验极佳，无更好的改进建议。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.