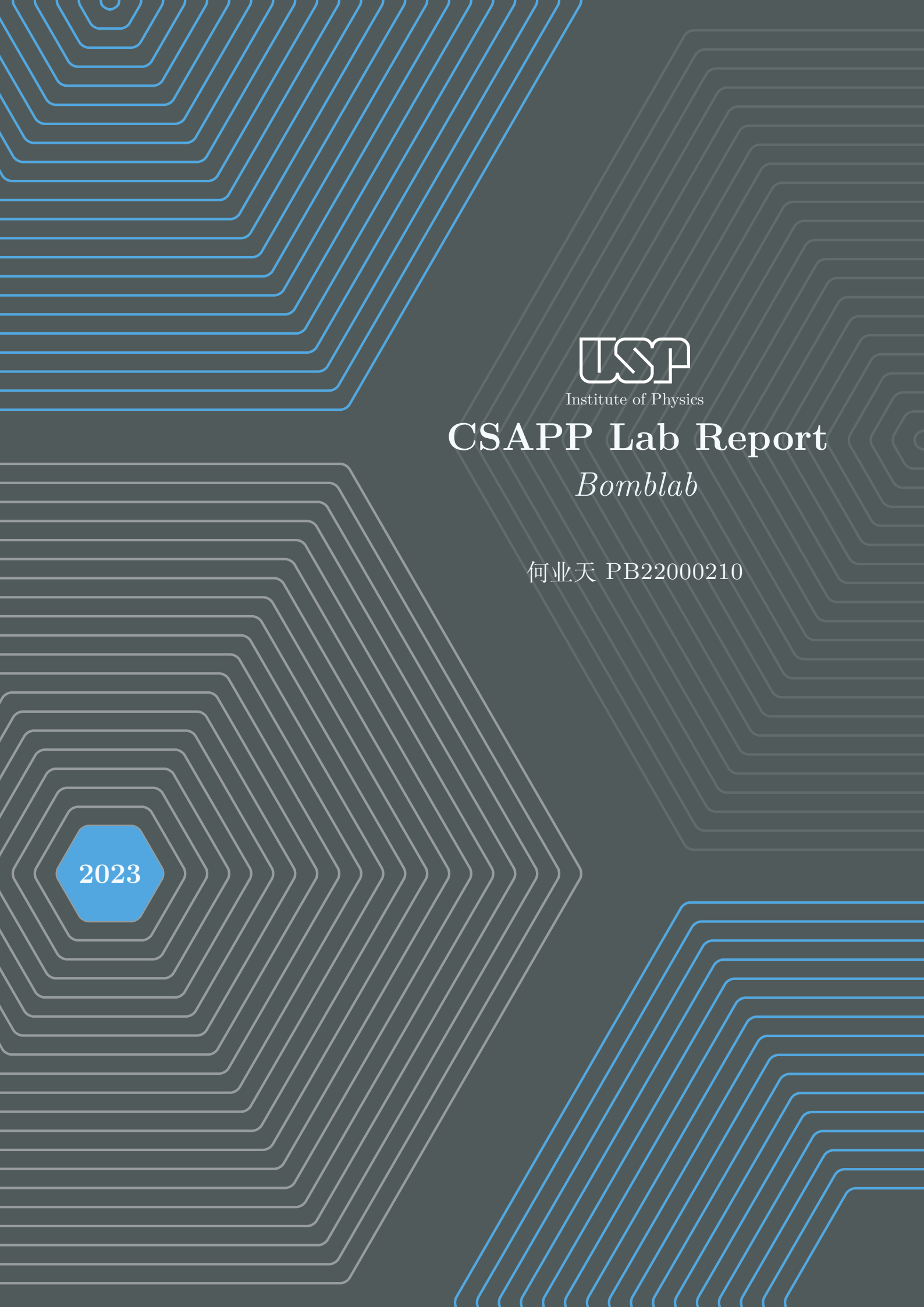


Institute of Physics

CSAPP Lab Report

Bomblab

何业天 PB22000210

2023

Phase_1 2
Phase_2 3
Phase_3 4
Phase_4 5
Phase_5 6
Phase_6 7

HOMEWORK N°

2

实验前的准备

1. 实验环境：同上一节 datalab 一样，通过虚拟机操作 Centos 7 系统。
2. 获取实验文件：在官网上的 Self-Study-Handout 上获取 bomb 文件，由 git 传入到虚拟机中的文件夹中。
3. 建立文件：创建了 ans.text 来储存每一个谜题的答案。利用命令 objdump bomb -d > my_bomb.txt 来存储 bomb 汇编代码，便于查看。
4. 运行方法：采用 gdb 进行调试，通过命令 gdb bomb 来调试 bomb 程序。进入调试后，输入 r 来运行程序，输入 break+ 地址值来添加断点，delete 来删除断点，输入 disas 来反汇编，通过 x 来查看地址内存中的值，通过 info 来查阅栈帧与寄存器数据。

Phase_1

我们先在 my_bomb.txt 里面找到 phase_1 函数，其汇编代码如下：

```
0000000000400ee0 <phase_1>:
  400ee0:       48 83 ec 08             sub    $0x8,%rsp
  400ee4:       be 00 24 40 00          mov    $0x402400,%esi
  400ee9:       e8 4a 04 00 00          callq  401338 <strings_not_equal>
  400eee:       85 c0                   test   %eax,%eax
  400ef0:       74 05                   je     400ef7 <phase_1+0x17>
  400ef2:       e8 43 05 00 00          callq  40143a <explode_bomb>
  400ef7:       48 83 c4 08             add    $0x8,%rsp
  400efb:       c3                      retq
```

这边一行行来分析：

首先将栈指针减去 8 字节，然后将地址 0x402400 赋值给第二个参数%esi，下一步是调用函数 <strings_not_equal> 我们找到该函数的汇编代码，其规格较长就不在这里展现了。可以观察到其调用的 <string_length> 函数

以及循环语句就是在判断两个字符串是否一模一样，最后根据返回值取值得知：倘若字符串不相同返回 1，相同则返回 0。并且我们从中知道，该函数是将第一个参数%rdi 的值与%rsi 的值进行比较，这里我们就明白了，实际上就是需要将我们输入的字符串与标准字符串比较。下一步的 test 就是在判断是否为 0，为 0 的话就跳过爆炸，成功解除炸弹。因此我们在 gdb 中利用 x/s 0x402400 来查看其代表的值，我们要输入的便是这个字符串。答案如下：

Border relations with Canada have never been better. (1.1)

Phase_2

```
0000000000400efc <phase_2>:
  400efc:       55                      push   %rbp
  400efd:       53                      push   %rbx
  400efe:       48 83 ec 28             sub    $0x28,%rsp
  400f02:       48 89 e6                mov    %rsp,%rsi
  400f05:       e8 52 05 00 00          callq  40145c <read_six_numbers>
  400f0a:       83 3c 24 01             cmpl   $0x1,(%rsp)
  400f0e:       74 20                   je     400f30 <phase_2+0x34>
  400f10:       e8 25 05 00 00          callq  40143a <explode_bomb>
  400f15:       eb 19                   jmp    400f30 <phase_2+0x34>
  400f17:       8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:       01 c0                   add    %eax,%eax
  400f1c:       39 03                   cmp    %eax,(%rbx)
  400f1e:       74 05                   je     400f25 <phase_2+0x29>
  400f20:       e8 15 05 00 00          callq  40143a <explode_bomb>
  400f25:       48 83 c3 04             add    $0x4,%rbx
  400f29:       48 39 eb                cmp    %rbp,%rbx
  400f2c:       75 e9                   jne    400f17 <phase_2+0x1b>
  400f2e:       eb 0c                   jmp    400f3c <phase_2+0x40>
  400f30:       48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f35:       48 8d 6c 24 18          lea    0x18(%rsp),%rbp
  400f3a:       eb db                   jmp    400f17 <phase_2+0x1b>
  400f3c:       48 83 c4 28             add    $0x28,%rsp
  400f40:       5b                      pop    %rbx
  400f41:       5d                      pop    %rbp
  400f42:       c3                      retq
```

首先压入两个被调用者保存的变量，下一步把减去 32 字节的栈指针地址赋给%rsi，调用 <read_six_numbers> 函数，我们不妨来看一下该函数：

```
000000000040145c <read_six_numbers>:
  40145c:       48 83 ec 18             sub    $0x18,%rsp
  401460:       48 89 f2                mov    %rsi,%rdx
  401463:       48 8d 4e 04             lea    0x4(%rsi),%rcx
  401467:       48 8d 46 14             lea    0x14(%rsi),%rax
  40146b:       48 89 44 24 08          mov    %rax,0x8(%rsp)
  401470:       48 8d 46 10             lea    0x10(%rsi),%rax
  401474:       48 89 04 24             mov    %rax,(%rsp)
  401478:       4c 8d 4e 0c             lea    0xc(%rsi),%r9
  40147c:       4c 8d 46 08             lea    0x8(%rsi),%r8
  401480:       be c3 25 40 00          mov    $0x4025c3,%esi
  401485:       b8 00 00 00 00          mov    $0x0,%eax
  40148a:       e8 61 f7 ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  40148f:       83 f8 05                cmp    $0x5,%eax
  401492:       7f 05                   jg     401499 <read_six_numbers+0x3d>
  401494:       e8 a1 ff ff ff          callq  40143a <explode_bomb>
  401499:       48 83 c4 18             add    $0x18,%rsp
  40149d:       c3                      retq
```

该函数主要是利用了 <soc99_sscanf@plt> 函数将输入的字符串转化成 6 个数，倘若不是六个就爆炸。在读入后，6 个数存储在%rsi 中，每个数占 4 字节的存储空间，一直分配到%rsi+18。接着是比较数组中的第一个元素是不是 1，如果是就不爆炸，反之爆炸。跳转到 400f30 后将%rbx 加 4 字节，也就是考虑下一个数。接下来的 400f17 到 400f2c 为一个循坏，在这个循环中寄存器%eax 中存储的是数组上一个数的两倍，判断两者是否相等，如果不等就爆炸。整个循环在数组到达最后一个元素后停止。由此我们可以知道我们要输入的数组需要：第一个元素为 1；数组为公比为 2 的等比数组。答案如下：

```
1  2  4  8  16  32
```

(1.2)

Phase_3

```
0000000000400f43 <phase_3>:
  400f43:       48 83 ec 18             sub    $0x18,%rsp
  400f47:       48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  400f4c:       48 8d 54 24 08          lea    0x8(%rsp),%rdx
  400f51:       be cf 25 40 00          mov    $0x4025cf,%esi
  400f56:       b8 00 00 00 00          mov    $0x0,%eax
  400f5b:       e8 90 fc ff ff          callq  400bf0 <__isoc99_sscanf@plt>
  400f60:       83 f8 01                cmp    $0x1,%eax
  400f63:       7f 05                   jg     400f6a <phase_3+0x27>
  400f65:       e8 d0 04 00 00          callq  40143a <explode_bomb>
  400f6a:       83 7c 24 08 07          cmpl   $0x7,0x8(%rsp)
  400f6f:       77 3c                   ja     400fad <phase_3+0x6a>
  400f71:       8b 44 24 08             mov    0x8(%rsp),%eax
  400f75:       ff 24 c5 70 24 40 00    jmpq   *0x402470(,%rax,8)
  400f7c:       b8 cf 00 00 00          mov    $0xcf,%eax
  400f81:       eb 3b                   jmp    400fbe <phase_3+0x7b>
  400f83:       b8 c3 02 00 00          mov    $0x2c3,%eax
  400f88:       eb 34                   jmp    400fbe <phase_3+0x7b>
  400f8a:       b8 00 01 00 00          mov    $0x100,%eax
  400f8f:       eb 2d                   jmp    400fbe <phase_3+0x7b>
  400f91:       b8 85 01 00 00          mov    $0x185,%eax
  400f96:       eb 26                   jmp    400fbe <phase_3+0x7b>
  400f98:       b8 ce 00 00 00          mov    $0xce,%eax
  400f9d:       eb 1f                   jmp    400fbe <phase_3+0x7b>
  400f9f:       b8 aa 02 00 00          mov    $0x2aa,%eax
  400fa4:       eb 18                   jmp    400fbe <phase_3+0x7b>
  400fa6:       b8 47 01 00 00          mov    $0x147,%eax
  400fab:       eb 11                   jmp    400fbe <phase_3+0x7b>
  400fad:       e8 88 04 00 00          callq  40143a <explode_bomb>
  400fb2:       b8 00 00 00 00          mov    $0x0,%eax
  400fb7:       eb 05                   jmp    400fbe <phase_3+0x7b>
  400fb9:       b8 37 01 00 00          mov    $0x137,%eax
  400fbe:       3b 44 24 0c             cmp    0xc(%rsp),%eax
  400fc2:       74 05                   je     400fc9 <phase_3+0x86>
  400fc4:       e8 71 04 00 00          callq  40143a <explode_bomb>
  400fc9:       48 83 c4 18             add    $0x18,%rsp
  400fcd:       c3                      retq
```

首先将%rcx 与%rdx 和栈指针挂钩，然后将 0x4025cf 赋给了%esi，用 x/s 0x4025cf 可以看出%esi 存储的是读取的两个数字，也就是说这题中我们需要输入两个数，如果不是两个数便会爆炸，这两个数分别对应着%rcx 与%rdx。下一步判断第一个数是否大于 7，如果大于则爆炸。从 400f75 到 400fb9 实际上就是一个 switch 语句。分别是不同%rcx 值（0 到 7）对应的值。最后判断该值是否等于%rdx 的值，如果是的，则成功。这里我们取第一个数为 0，那么第二个数就是 0xcf 即 207。其他情况类似，就不一一计算了。

以下为答案（7 个中的 1 个）：

0 207 (1.3)

Phase_4

```
000000000040100c <phase_4>:
  40100c:       48 83 ec 18             sub    $0x18,%rsp
  401010:       48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
  401015:       48 8d 54 24 08          lea    0x8(%rsp),%rdx
  40101a:       be cf 25 40 00          mov    $0x4025cf,%esi
  40101f:       b8 00 00 00 00          mov    $0x0,%eax
  401024:       e8 c7 fb ff ff          callq  400bf0 <__isoc99_sscanf@plt
  401029:       83 f8 02                cmp    $0x2,%eax
  40102c:       75 07                   jne    401035 <phase_4+0x29>
  40102e:       83 7c 24 08 0e          cmpl   $0xe,0x8(%rsp)
  401033:       76 05                   jbe    40103a <phase_4+0x2e>
  401035:       e8 00 04 00 00          callq  40143a <explode_bomb>
  40103a:       ba 0e 00 00 00          mov    $0xe,%edx
  40103f:       be 00 00 00 00          mov    $0x0,%esi
  401044:       8b 7c 24 08             mov    0x8(%rsp),%edi
  401048:       e8 81 ff ff ff          callq  400fce <func4>
  40104d:       85 c0                   test   %eax,%eax
  40104f:       75 07                   jne    401058 <phase_4+0x4c>
  401051:       83 7c 24 0c 00          cmpl   $0x0,0xc(%rsp)
  401056:       74 05                   je     40105d <phase_4+0x51>
  401058:       e8 dd 03 00 00          callq  40143a <explode_bomb>
  40105d:       48 83 c4 18             add    $0x18,%rsp
  401061:       c3                      retq
```

可以看到前几步和 phase_3 如出一辙，就是读取两个数并存储在%rcx 与%rdx 中。当第一个数小于等于 14 时，先对几个参数初始化，%edx 赋值为 14，%esi 赋值为 0，%edi 赋值为第一个数。下一步就是调用 <func4> 函数了，也是本题的核心，来看一下该函数的代码：

```
0000000000400fce <func4>:
  400fce:       48 83 ec 08             sub    $0x8,%rsp
  400fd2:       89 d0                   mov    %edx,%eax
  400fd4:       29 f0                   sub    %esi,%eax
  400fd6:       89 c1                   mov    %eax,%ecx
  400fd8:       c1 e9 1f                shr    $0x1f,%ecx
  400fdb:       01 c8                   add    %ecx,%eax
  400fdd:       d1 f8                   sar    %eax
  400fdf:       8d 0c 30                lea    (%rax,%rsi,1),%ecx
  400fe2:       39 f9                   cmp    %edi,%ecx
  400fe4:       7e 0c                   jle    400ff2 <func4+0x24>
  400fe6:       8d 51 ff                lea    -0x1(%rcx),%edx
  400fe9:       e8 e0 ff ff ff          callq  400fce <func4>
  400fee:       01 c0                   add    %eax,%eax
  400ff0:       eb 15                   jmp    401007 <func4+0x39>
  400ff2:       b8 00 00 00 00          mov    $0x0,%eax
  400ff7:       39 f9                   cmp    %edi,%ecx
  400ff9:       7d 0c                   jge    401007 <func4+0x39>
  400ffb:       8d 71 01                lea    0x1(%rcx),%esi
  400ffe:       e8 cb ff ff ff          callq  400fce <func4>
  401003:       8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
  401007:       48 83 c4 08             add    $0x8,%rsp
  40100b:       c3                      retq
```

一开始将%eax 赋值为了%edx 与%esi 的差值，接下来对%eax 要进行系列操作，较为复杂，具体算式如下：eax = (eax + eax ≫ 31) ≫ 1，下一步是给%ecx 赋值，简化计算结果可以知道%ecx=(%edx+%esi)/2。下一步便是进行递归操作了，假如%ecx=%edi 时，函数返回 0；小于时，则进行迭代，

迭代的参数中%edx 与%edi 不变，然后%esi 变为%ecx-1，函数返回迭代两倍 func4 返回值加 1；大于时，%edi 与%esi 不变，然后%edx 变为%ecx-1,，函数返回迭代两倍 func4 返回值。

func4 函数返回值后，假如返回值为 0，并且第二个数为 0，则成功拆弹。那么第一个数是什么呢？显然第一次迭代假如相等，则第一个数就是 7，这便是第一个解。假如第一个数大于或小于 7，找到与第一个数相等的数类似于二分查找，为了能使最终返回值等于 0，我们不能使第一个数大于匹配数，因此其他满足调节的数为 0，1，3。以下为 4 组答案：

(0/1/3/7)　0　　(1.4)

Phase_5

```
0000000000401062 <phase_5>:
  401062:       53                      push   %rbx
  401063:       48 83 ec 20             sub    $0x20,%rsp
  401067:       48 89 fb                mov    %rdi,%rbx
  40106a:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  401071:       00 00
  401073:       48 89 44 24 18          mov    %rax,0x18(%rsp)
  401078:       31 c0                   xor    %eax,%eax
  40107a:       e8 9c 02 00 00          callq  40131b <string_length>
  40107f:       83 f8 06                cmp    $0x6,%eax
  401082:       74 4e                   je     4010d2 <phase_5+0x70>
  401084:       e8 b1 03 00 00          callq  40143a <explode_bomb>
  401089:       eb 47                   jmp    4010d2 <phase_5+0x70>
  40108b:       0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
  40108f:       88 0c 24                mov    %cl,(%rsp)
  401092:       48 8b 14 24             mov    (%rsp),%rdx
  401096:       83 e2 0f                and    $0xf,%edx
  401099:       0f b6 92 b0 24 40 00    movzbl 0x4024b0(%rdx),%edx
  4010a0:       88 54 04 10             mov    %dl,0x10(%rsp,%rax,1)
  4010a4:       48 83 c0 01             add    $0x1,%rax
  4010a8:       48 83 f8 06             cmp    $0x6,%rax
  4010ac:       75 dd                   jne    40108b <phase_5+0x29>
  4010ae:       c6 44 24 16 00          movb   $0x0,0x16(%rsp)
  4010b3:       be 5e 24 40 00          mov    $0x40245e,%esi
  4010b8:       48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  4010bd:       e8 76 02 00 00          callq  401338 <strings_not_equal>
  4010c2:       85 c0                   test   %eax,%eax
  4010c4:       74 13                   je     4010d9 <phase_5+0x77>
  4010c6:       e8 6f 03 00 00          callq  40143a <explode_bomb>
  4010cb:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
  4010d0:       eb 07                   jmp    4010d9 <phase_5+0x77>
  4010d2:       b8 00 00 00 00          mov    $0x0,%eax
  4010d7:       eb b2                   jmp    40108b <phase_5+0x29>
  4010d9:       48 8b 44 24 18          mov    0x18(%rsp),%rax
  4010de:       64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  4010e5:       00 00
  4010e7:       74 05                   je     4010ee <phase_5+0x8c>
  4010e9:       e8 42 fa ff ff          callq  400b30 <__stack_chk_fail@plt>
  4010ee:       48 83 c4 20             add    $0x20,%rsp
  4010f2:       5b                      pop    %rbx
  4010f3:       c3                      retq
```

一开始注意一下，40106a 到 401073 是设置了金丝雀值来保护栈，下一步便将%eax 值清零了。接着判断输入的字符是否为 6，不是就爆炸。%rax 清零后进入循环语句。接下来的一段有点绕，先来说明一下整个循环语句的结果：其为 for 语句，%rax 初始值为 0，接着每一次加 1，直到其等于 6 终止。

下面详细讲一下循环内的操作。首先%ecx=%rbx+%rax，其实这里的%rbx 就是字符串第一个字符所在的地址，%ecx 则代表了字符串的第%rax+1 个字符。下一步利用中间变量将%rdx 赋值为%ecx，并且%edx 与 0xf 进行与操作，也就是取其后 4 位的值。接着将 $edx 再赋值为%rdx+0x4024b0 的地址值，其实就是将数组设定在 0x4024b0 处，0x4024b0 代表的字符串如下：

```
(gdb) x/s 0x4024b0
0x4024b0 <array.3449>:  "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
```

注意上述的%rdx 肯定是在 0 到 15 范围内的，即取到的字符肯定是上述字符串的前 16 个字符“maduiersnfotvbyl”，其实实现的效果也就是用%rdx 来选择该字符串中的内容，并且循环六次。下一步将%rsp+0x16 赋值为 0，实际就是给这个六个字符加上一个字符结尾符。接着给%esi 赋值为 0x40245e，通过 x/s 操作可以得到其中的字符串为”flyers”。后面就好说了，实际上就是调用字符串匹配函数，要取的字符串与” flyers” 相同。在长字符串中找到这六个字母对应的位置：9、15、14、5、6、7。上述为输入字符和 0xf 进行与操作的结果。我们查询 ASCII 码表可以得到：9、5、6、7 取这四个数字本身即可；15 和 14 就取”? ” 和”>”。这便得出了许多答案中的一组：

$$9 \quad ? \quad > \quad 5 \quad 6 \quad 7 \tag{1.5}$$

Phase_6

phase_6 是挑战性最大的一个谜题，我自己也是被其中的一些代码理解卡住了很久。因为代码较长，就分块来看。

```
00000000004010f4 <phase_6>:
  4010f4:       41 56                   push   %r14
  4010f6:       41 55                   push   %r13
  4010f8:       41 54                   push   %r12
  4010fa:       55                      push   %rbp
  4010fb:       53                      push   %rbx
  4010fc:       48 83 ec 50             sub    $0x50,%rsp
  401100:       49 89 e5                mov    %rsp,%r13
  401103:       48 89 e6                mov    %rsp,%rsi
  401106:       e8 51 03 00 00          callq  40145c <read_six_numbers>
  40110b:       49 89 e6                mov    %rsp,%r14
  40110e:       41 bc 00 00 00 00       mov    $0x0,%r12d
  401114:       4c 89 ed                mov    %r13,%rbp
  401117:       41 8b 45 00             mov    0x0(%r13),%eax
  40111b:       83 e8 01                sub    $0x1,%eax
  40111e:       83 f8 05                cmp    $0x5,%eax
  401121:       76 05                   jbe    401128 <phase_6+0x34>
  401123:       e8 12 03 00 00          callq  40143a <explode_bomb>
  401128:       41 83 c4 01             add    $0x1,%r12d
  40112c:       41 83 fc 06             cmp    $0x6,%r12d
  401130:       74 21                   je     401153 <phase_6+0x5f>
```

前面这些内容比较简单，基本就是赋初值以及读入 6 个数字，下面可以看出有一个 for 循环语句，%r12 其实就是一个类似于 i 的因子，每次循环 +1，到达 6 时停止。

```
401132:       44 89 e3                 mov    %r12d,%ebx
401135:       48 63 c3                 movslq %ebx,%rax
401138:       8b 04 84                 mov    (%rsp,%rax,4),%eax
40113b:       39 45 00                 cmp    %eax,0x0(%rbp)
40113e:       75 05                    jne    401145 <phase_6+0x51>
401140:       e8 f5 02 00 00           callq  40143a <explode_bomb>
401145:       83 c3 01                 add    $0x1,%ebx
401148:       83 fb 05                 cmp    $0x5,%ebx
40114b:       7e e8                    jle    401135 <phase_6+0x41>
40114d:       49 83 c5 04              add    $0x4,%r13
401151:       eb c1                    jmp    401114 <phase_6+0x20>
```

接下来是一个内层循环，将%ebx 的初值赋为%r12，然后每次循环 +1，直到等于 5。中间一段内容将%eax 赋值为 (%rsp+%rax*4) 对应的值，并且比较该值是否与%rbq 值相等，相等的话就爆炸。%rbq 在这里是%r13 对应的值。内层循环结束后，%r13+4。从这里分析，可以直到%rsp 中存储的应该是数组的地址信息，并且对于这个数组中的数，两两不相等，并且这几个数都需要在 1 到 6 之间，综合可知我们就是要以一个特定的顺序输入 1 到 6。

```
401153:       48 8d 74 24 18           lea    0x18(%rsp),%rsi
401158:       4c 89 f0                 mov    %r14,%rax
40115b:       b9 07 00 00 00           mov    $0x7,%ecx
401160:       89 ca                    mov    %ecx,%edx
401162:       2b 10                    sub    (%rax),%edx
401164:       89 10                    mov    %edx,(%rax)
401166:       48 83 c0 04              add    $0x4,%rax
40116a:       48 39 f0                 cmp    %rsi,%rax
40116d:       75 f1                    jne    401160 <phase_6+0x6c>
```

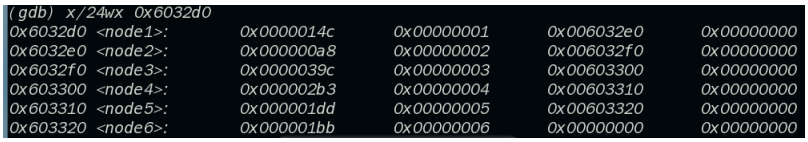
接下来是一个独立的 for 语句。首先初值上，%rsi 赋值为%rsp+0x18,%ecx 赋值为 7，而%rax 赋值为%r14 即%rsp，即为%rax 设定了初值，接下来每一次循环%rax+4，直到%rax 到达%rsi 时停止。循环的内容实际上就是使%rsp 指向的数组中的每个数变成 7 与它的差值。

```
40116f:       be 00 00 00 00           mov    $0x0,%esi
401174:       eb 21                    jmp    401197 <phase_6+0xa3>
401176:       48 8b 52 08              mov    0x8(%rdx),%rdx
40117a:       83 c0 01                 add    $0x1,%eax
40117d:       39 c8                    cmp    %ecx,%eax
40117f:       75 f5                    jne    401176 <phase_6+0x82>
401181:       eb 05                    jmp    401188 <phase_6+0x94>
401183:       ba d0 32 60 00           mov    $0x6032d0,%edx
401188:       48 89 54 74 20           mov    %rdx,0x20(%rsp,%rsi,2)
40118d:       48 83 c6 04              add    $0x4,%rsi
401191:       48 83 fe 18              cmp    $0x18,%rsi
401195:       74 14                    je     4011ab <phase_6+0xb7>
401197:       8b 0c 34                 mov    (%rsp,%rsi,1),%ecx
40119a:       83 f9 01                 cmp    $0x1,%ecx
40119d:       7e e4                    jle    401183 <phase_6+0x8f>
40119f:       b8 01 00 00 00           mov    $0x1,%eax
4011a4:       ba d0 32 60 00           mov    $0x6032d0,%edx
4011a9:       eb cb                    jmp    401176 <phase_6+0x82>
```

这一段又是有内外循环，也是本谜题的核心。先来看大循环，这里一开始将%rsi 赋值为 0 作为初值，接着每个循环使其加 4，直到其等于 0x18。在循坏里，将%ecx 赋值为%rsp+%rsi，假如%ecx 小于等于 1 时，将%eax 设为 1，%edx 设为 0x6032d0。我们来看一下 0x6032d0 有没有什么信息：

```
(gdb) x/s 0x6032d0
0x6032d0 <node1>:       "L\001"
```

这里的 node 就有点提示的味道了，0x6032d0 显然是一个与上述地址距离很大的区域，可以看出这里将%edx 赋值到了一块新的内存空间。下面便是一个内循环，将%eax 的初值设为 1，每次循坏加 1，直到等于%ecx (这里可以看出是内循环)，内循环中每次将%rdx 加 8。内循环结束之后。将%rsp+2*%rsi+0x20 赋值为%rdx。从这一系列操作中我们可以意识到，这实际上是一个建立链表的操作。根据网络上的方法，我们用指令 x/24wx 0x6032d0 来展现一下信息：



可以看出来确实是一个单向链表，每一个节点指向下一个节点，前 4 字节为存储的数值，4—8 字节为索引值，8—12 位为指向的地址。结点 1 到 6 分别存储的数值为：332、168、924、691、477、443。我们输入的数组实际上是代表着索引值的意思，输入的也就是将该链表重新排序后的顺序。并且在%rsp+0x20 出建立一个新的指针数组去存储指向结点的指针。

```
4011ab:   48 8b 5c 24 20          mov    0x20(%rsp),%rbx
4011b0:   48 8d 44 24 28          lea    0x28(%rsp),%rax
4011b5:   48 8d 74 24 50          lea    0x50(%rsp),%rsi
4011ba:   48 89 d9                mov    %rbx,%rcx
4011bd:   48 8b 10                mov    (%rax),%rdx
4011c0:   48 89 51 08             mov    %rdx,0x8(%rcx)
4011c4:   48 83 c0 08             add    $0x8,%rax
4011c8:   48 39 f0                cmp    %rsi,%rax
4011cb:   74 05                   je     4011d2 <phase_6+0xde>
4011cd:   48 89 d1                mov    %rdx,%rcx
4011d0:   eb eb                   jmp    4011bd <phase_6+0xc9>
4011d2:   48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx)
4011d9:   00
```

再接着又是一个循环，一开始是对初值的设定。首先将%rbx 设为%rsp+0x20, 也就是存储结点指针的数组，将%rcx 赋值为%rbx，并将%rax 初值设定为%rsp+0x28, 每次循环加 8，直到到达%rsp+0x50。循环的操作实际上就是对这个新的指针进行串联。

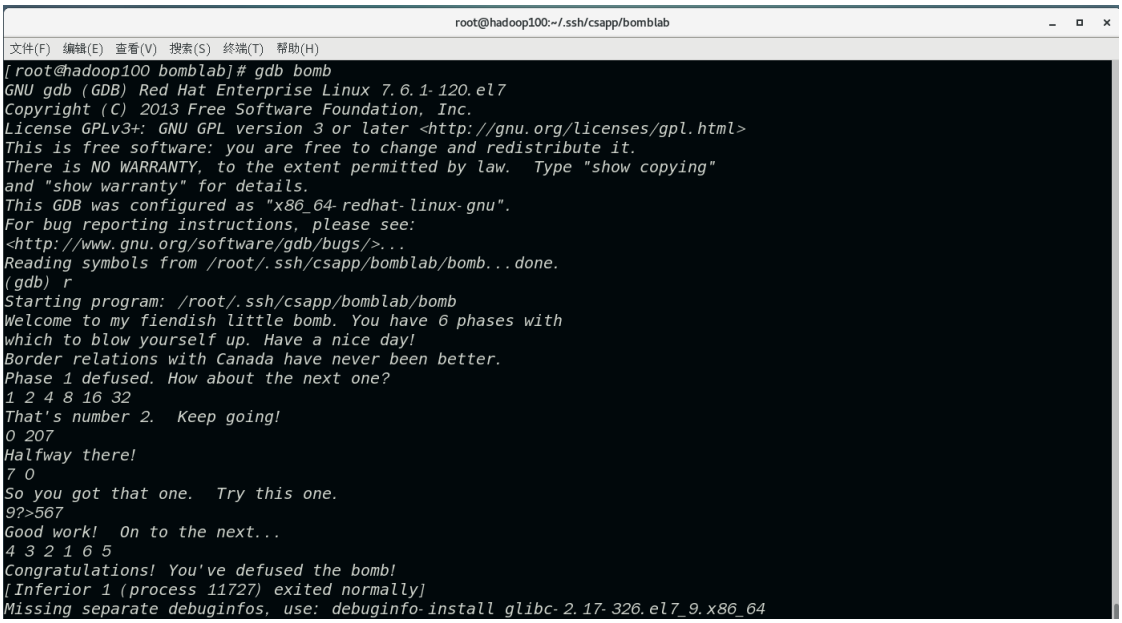
```
4011da:   bd 05 00 00 00          mov    $0x5,%ebp
4011df:   48 8b 43 08             mov    0x8(%rbx),%rax
4011e3:   8b 00                   mov    (%rax),%eax
4011e5:   39 03                   cmp    %eax,(%rbx)
4011e7:   7d 05                   jge    4011ee <phase_6+0xfa>
4011e9:   e8 4c 02 00 00          callq  40143a <explode_bomb>
4011ee:   48 8b 5b 08             mov    0x8(%rbx),%rbx
4011f2:   83 ed 01                sub    $0x1,%ebp
4011f5:   75 e8                   jne    4011df <phase_6+0xeb>
4011f7:   48 83 c4 50             add    $0x50,%rsp
4011fb:   5b                      pop    %rbx
4011fc:   5d                      pop    %rbp
4011fd:   41 5c                   pop    %r12
4011ff:   41 5d                   pop    %r13
401201:   41 5e                   pop    %r14
401203:   c3                      retq
```

最后一个循环便是对指针数组的最后一个限制条件。将%rax 设为%rbx+8，这个循环将%ebp 设置初值为 5，每次循环减一，直到其等于 0 时停止。循环内的操作比较好理解，实际上就是该指针数组的下一个结点数值不能大于上一个的，换一句话说就是递减。

综上，我们来反推输入数组，首先根据结点数值排成递减数列，可以得到顺序为：3、4、5、6、1、2。这便是经过与 7 作差后的数组值，反向操作后得到顺序应为：4、3、2、1、6、5。即答案为：

$$4\quad 3\quad 2\quad 1\quad 6\quad 5 \tag{1.6}$$

实验运行结果



(在看程序汇编代买时注意到了 func7 和 secret_phase，可惜没有找到运行的方式，便没有下一步的分析。)

实验总结

bomblab 的设计十分有意思，并且其每一个谜题其实也代表了一类汇编代码的运用情形，将这些情形转化为 c 语言后可能会十分简单，但汇编更可以展现出来存储空间的细微变化。谜题 1 到 6 也是循序渐进的展现了不同的主题：第一个是熟悉汇编语言和函数调用；第二个是熟悉判断语句与循环语句；第三个是 switch 语句的运用；第四个是懂得迭代函数（二分法迭代）；第五个熟悉是数组的建立与使用；第六个最有挑战性，是之前重点的集合，并加上了链表结构。做完这六个实验，在阅读汇编代码层面上基本没有太大问题了，并且对内存空间的分配也有了更加清晰的认识。