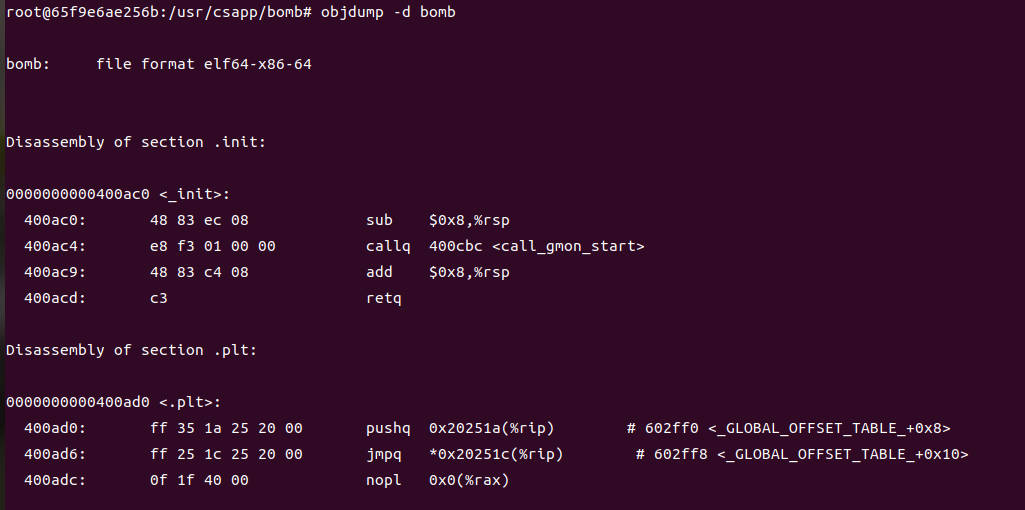
要查看机器代码文件的内容，有一类称为反汇编器（disassembler，assembler是汇编程序，dis-加在某些词语前表示相反的意思）的程序非常有用。这些程序根据机器代码产生一种类似于汇编代码的格式。在linux系统中，带‘-d’命令行标志的程序OBJDUMP（表示“object dump”）可以充当这个角色。

反汇编器只是基于机器代码文件中的字节序列来确定汇编代码。它不需要访问该程序的源代码或汇编代码。

反汇编器使用的指令命名规则与GCC生成的汇编代码使用的有些细微的差别。很多指令结尾的‘q’被省略了。这些后缀是大小指示符，在大多数情况中可以省略。相反，反汇编器给call和ret指令添加了‘q’后缀，同样省略这些后缀也没有关系。

**反汇编bomb文件**

objdump -d bomb

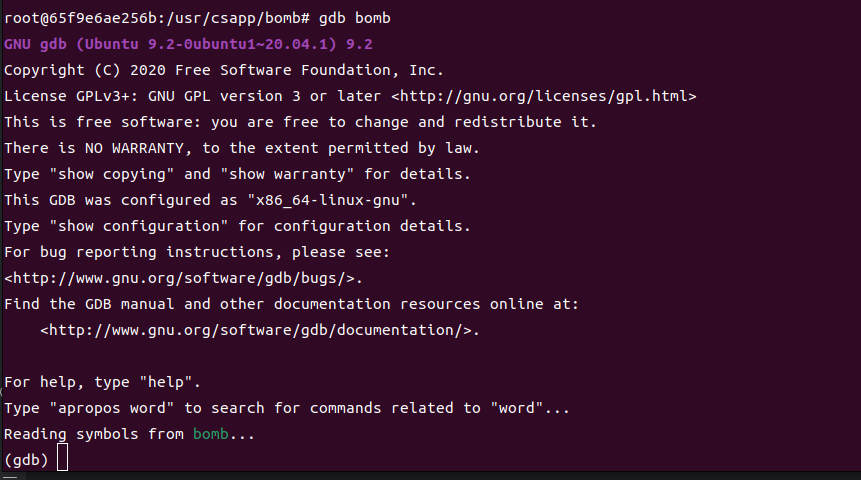


GUN的调试器GDB提供了许多有用的特性，支持机器级程序的运行时评估和分析。

通常的方法是在程序中感兴趣的地方附近设置断点。断点可以设置在函数入口后面，或是一个程序的地址处。程序在执行过程中遇到一个断点时，程序会停下来，并将控制返回给用户。在断点处，我们能够以各种方式查看各个寄存器和内存位置。我们也可以单步跟踪程序，一次只执行几条指令，或是前进到下一个断点。

**使用GDB调试器**

gdb bomb



docker容器内使用gdb有问题可以参照[在Docker内部使用gdb调试器报错-Operation not permitted](note://WEBd5cf57d3a5d3e242166e3040b1bbb34d)

gdb命令可以参照书本P194

**解题步骤**

**phase\_1**

**答案：**"Border relations with Canada have never been better."

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> #如果返回值等于0,则跳转至400ef7,输入字符和目标字符不等时，返回值为1,否则返回值为0

400ef2:e8 43 05 00 00 callq 40143a <explode\_bomb>

400ef7:48 83 c4 08 add $0x8,%rsp

400efb:c3 retq

string\_length过程目的是计算字符的个数，在gdb中使用print \* (char \*) 0x603780，可以确定main过程中phase\_1之前的read\_line的返回值0x603780是输入字符的首地址

000000000040131b <string\_length>:

40131b:80 3f 00 cmpb $0x0,(%rdi)

40131e:74 12 je 401332 <string\_length+0x17>

401320:48 89 fa mov %rdi,%rdx

401323:48 83 c2 01 add $0x1,%rdx

401327:89 d0 mov %edx,%eax

401329:29 f8 sub %edi,%eax

40132b:80 3a 00 cmpb $0x0,(%rdx) #寄存器外面有括号，等于是访问寄存器里的值指向的存储器位置，即访问存储器

40132e:75 f3 jne 401323 <string\_length+0x8>

401330:f3 c3 repz retq

401332:b8 00 00 00 00 mov $0x0,%eax

401337:c3 retq

phase\_1需要判断输入字符和目标字符是否相等，目标字符的首地址为0x402400，长度为0x34，使用"print \* (char \*) 0x402400"、"print \* (char \*) 0x402401"直到"print \* (char\*) 0x402433"，可以打印出目标字符，而这正是需要的答案

0000000000401338 <strings\_not\_equal>:

401338:41 54 push %r12

40133a:55 push %rbp

40133b:53 push %rbx

40133c:48 89 fb mov %rdi,%rbx #0x603780 这是存储器中输入的字符的首地址

40133f:48 89 f5 mov %rsi,%rbp #0x402400 这是存储器中目标字符的首地址

401342:e8 d4 ff ff ff callq 40131b <string\_length> #计算有多少个字符

401347:41 89 c4 mov %eax,%r12d #输入字符的长度

40134a:48 89 ef mov %rbp,%rdi #0x402400

40134d:e8 c9 ff ff ff callq 40131b <string\_length> #计算存储器中首地址为0x402400的字符的长度为0x34

401352:ba 01 00 00 00 mov $0x1,%edx

401357:41 39 c4 cmp %eax,%r12d #比较输入字符和首地址为0x402400的字符的大小

40135a:75 3f jne 40139b <strings\_not\_equal+0x63>

40135c:0f b6 03 movzbl (%rbx),%eax #取出一个输入字符

40135f:84 c0 test %al,%al #判断输入字符是否等于0

401361:74 25 je 401388 <strings\_not\_equal+0x50> #等于0则跳转至0x401388

401363:3a 45 00 cmp 0x0(%rbp),%al #判断输入字符和目标字符对应位置上的字符是否相等

401366:74 0a je 401372 <strings\_not\_equal+0x3a> #相等则跳转至0x401372

401368:eb 25 jmp 40138f <strings\_not\_equal+0x57>

40136a:3a 45 00 cmp 0x0(%rbp),%al #判断输入字符和目标字符对应位置上的字符是否相等，从第2个字符开始，一直在这里循环

40136d:0f 1f 00 nopl (%rax) #nopl 用于在指令流中插入空闲nop指令，使指令之后的访问可以对齐

401370:75 24 jne 401396 <strings\_not\_equal+0x5e> #不相等则跳转至0x401396

401372:48 83 c3 01 add $0x1,%rbx #指向输入字符的指针加1

401376:48 83 c5 01 add $0x1,%rbp #指向目标字符的指针加1

40137a:0f b6 03 movzbl (%rbx),%eax #取出一个输入字符

40137d:84 c0 test %al,%al #判断输入字符是否等于0

40137f:75 e9 jne 40136a <strings\_not\_equal+0x32> #不相等则跳转至0x40136a

401381:ba 00 00 00 00 mov $0x0,%edx

401386:eb 13 jmp 40139b <strings\_not\_equal+0x63> #如果输入字符等于目标字符，最后会将%edx设置为0,并跳转至40139b

401388:ba 00 00 00 00 mov $0x0,%edx

40138d:eb 0c jmp 40139b <strings\_not\_equal+0x63>

40138f:ba 01 00 00 00 mov $0x1,%edx

401394:eb 05 jmp 40139b <strings\_not\_equal+0x63>

401396:ba 01 00 00 00 mov $0x1,%edx

40139b:89 d0 mov %edx,%eax #如果输入字符和目标字符不同，返回值%eax=1；反之，如果相同，%eax=0

40139d:5b pop %rbx

40139e:5d pop %rbp

40139f:41 5c pop %r12

4013a1:c3 retq

**phase\_2**

**答案：**"1 2 4 8 16 32"

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 #%rsi的值设置为栈指针

400f05:e8 52 05 00 00 callq 40145c <read\_six\_numbers> #判断输入字符前6个是否是数字,若是，返回值为0x6,此时的帧如图4所示

400f0a:83 3c 24 01 cmpl $0x1,(%rsp) #将R[%rsp]指向的存储器值与1进行比较

400f0e:74 20 je 400f30 <phase\_2+0x34> #相等则跳转至0x400f30

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 #取出R[%rbx-0x4]指向的值,即R[%rsp]指向的值，保存在%eax中,一直在此循环

400f1a:01 c0 add %eax,%eax #eax的值\*2

400f1c:39 03 cmp %eax,(%rbx) #eax和R[%rbx]指向的值进行比较

400f1e:74 05 je 400f25 <phase\_2+0x29> #如果相等，则跳转至0x400f25

400f20:e8 15 05 00 00 callq 40143a <explode\_bomb>

400f25:48 83 c3 04 add $0x4,%rbx #%rbx中的值+0x4，即R[%rbx]指向R[%rsp]+0x8

400f29:48 39 eb cmp %rbp,%rbx #比较R[%rbp]和R[%rbx]的值

400f2c:75 e9 jne 400f17 <phase\_2+0x1b> #如果不相等，则跳转至0x400f17，一直在此循环,直到R[%rbx]=R[%rsp]+0x18

400f2e:eb 0c jmp 400f3c <phase\_2+0x40>

400f30:48 8d 5c 24 04 lea 0x4(%rsp),%rbx #%rbx指向(R[%rsp]+0x4)

400f35:48 8d 6c 24 18 lea 0x18(%rsp),%rbp #%rbp指向(R[%rsp]+0x18)

400f3a:eb db jmp 400f17 <phase\_2+0x1b>

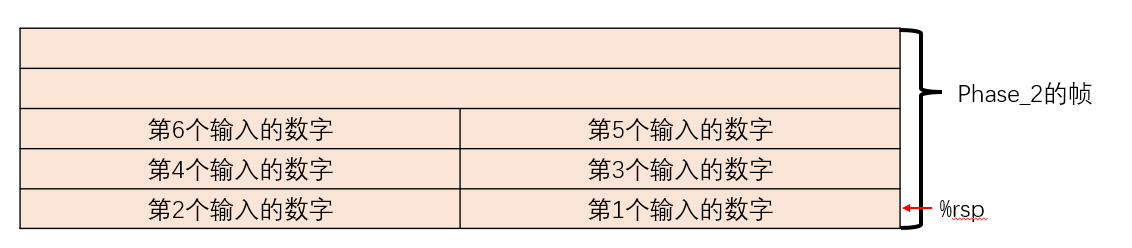
400f3c:48 83 c4 28 add $0x28,%rsp

400f40:5b pop %rbx

400f41:5d pop %rbp

400f42:c3 retq

phase\_2中的read\_six\_numbers目的是读取输入字符的前6个字符，判断是不是数字字符，并设置phase\_2的帧下图。



通过print \* (int \*) R[%rsp]可以确定R[%rsp]指向的是第一个输入的数字。

(gdb) print /x $rsp

$117 = 0x7fffffffe600

(gdb) print \* (int \*) 0x7fffffffe600

$118 = 1

**phase\_3**

**答案：**"0 207"，"1 311"，"2 707"，"3 256"，"4 389"，"5 206"，"6 682"，"7 327"

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 #查看%esi指向的字符为%d,可以猜到需要输入字符为数字

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 #返回值%eax表示输入字符中的数字的个数

400f63:7f 05 jg 400f6a <phase\_3+0x27> #如果个数大于1,则跳转至0x400f6a

400f65:e8 d0 04 00 00 callq 40143a <explode\_bomb>

400f6a:83 7c 24 08 07 cmpl $0x7,0x8(%rsp) #比较(R[%rsp]+0x8)指向的值和0x7

400f6f:77 3c ja 400fad <phase\_3+0x6a> #如果大于0x7,则跳转至0x400fad，引爆炸弹，可知输入的第一个数字要小于或等于0x7

400f71:8b 44 24 08 mov 0x8(%rsp),%eax #取出(R[%rsp]+0x8)指向的值，保存在%eax中

400f75:ff 24 c5 70 24 40 00 jmpq \*0x402470(,%rax,8) #jmp \*0x402470(,%rax,8) means jmp \*(0x402470(,%rax,8)) and \* means the data at the address

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 #把0x137保存在%eax中

400fbe:3b 44 24 0c cmp 0xc(%rsp),%eax #比较(R[%rsp]+0xc)指向的值和%eax中值

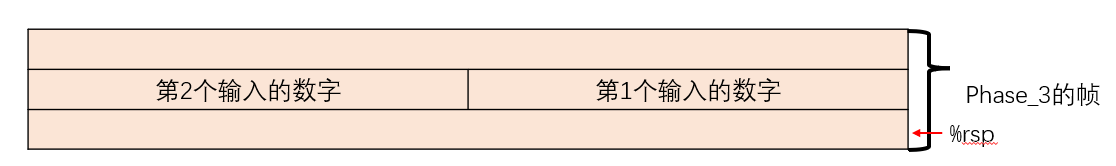
400fc2:74 05 je 400fc9 <phase\_3+0x86> #如果相等，则跳转至0x400fc9

400fc4:e8 71 04 00 00 callq 40143a <explode\_bomb>

400fc9:48 83 c4 18 add $0x18,%rsp

400fcd:c3 retq

callq 400bf0 <\_\_isoc99\_sscanf@plt>后phase\_3的帧如下图所示



phase\_3中的jmpq \*0x402470(,%rax,8)意思是 jmp \*(0x402470(,%rax,8)) and \* means the data at the address，即跳转到(0x402470(,%rax,8))指向的存储器的值指向的地址。目标地址和%rax中的值有关，也就是说，和输入数据的第一个数字有关，使用命令查看存储器中地址可以找到具体的跳转地址和第一个数字的关系，进而确定第二个数字。

(gdb) x/2g 0x402470

0x402470:0x0000000000400f7c0x0000000000400fb9

(gdb) x/2g 0x402478

0x402478:0x0000000000400fb90x0000000000400f83

(gdb) x/2g 0x402480

0x402480:0x0000000000400f830x0000000000400f8a

(gdb) x/2g 0x402488

0x402488:0x0000000000400f8a0x0000000000400f91

(gdb) x/2g 0x402490

0x402490:0x0000000000400f910x0000000000400f98

(gdb) x/2g 0x402498

0x402498:0x0000000000400f980x0000000000400f9f

(gdb) x/2g 0x4024a0

0x4024a0:0x0000000000400f9f0x0000000000400fa6

(gdb) x/2g 0x4024a8

0x4024a8:0x0000000000400fa60x737265697564616d

**phase\_4**

**答案：**"7 0"

由mov $0x4025cf,%esi和callq 400bf0 [\_\_isoc99\_sscanf@plt](http://__isoc99_sscanf@plt) 判断需要输入数字

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 #逻辑右移31位，结果为0x0

400fdb:01 c8 add %ecx,%eax #R[%eax]+R[%ecx],结果保存在%eax,由于%ecx中的值为0,相加的结果还是0xe

400fdd:d1 f8 sar %eax #算术右移一位，结果为0x7

400fdf:8d 0c 30 lea (%rax,%rsi,1),%ecx #计算R[%rax]+R[%rsi]\*1,结果为0x7,保存在%ecx中

400fe2:39 f9 cmp %edi,%ecx #比较%edi和%ecx中的值，即第一个数字和0x7

400fe4:7e 0c jle 400ff2 <func4+0x24> #如果大于或等于0x7，则跳转至0x400ff2,正确答案需要跳转

400fe6:8d 51 ff lea -0x1(%rcx),%edx #%edx的值-0x1，第一执行的结果为0x6

400fe9:e8 e0 ff ff ff callq 400fce <func4> #跳转至0x400fce，减少%ecx的值，直到%edi的值大于或等于%ecx中的值,跳转至0x400ff2

400fee:01 c0 add %eax,%eax

400ff0:eb 15 jmp 401007 <func4+0x39>

400ff2:b8 00 00 00 00 mov $0x0,%eax #返回值%rax设置为0x0,正确答案需要这样设置

400ff7:39 f9 cmp %edi,%ecx #比较%edi和%ecx中的值，即第一个数字和0x7

400ff9:7d 0c jge 401007 <func4+0x39> #如果小于或等于%ecx中的值，则跳转至0x401007，正确答案需要跳转

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

func4中0x400fe4要求第一个字符大于或等于0x7，0x400ff9要求第一个字符小于或等于0x7,所以第一个字符为0x7，phase\_4的0x401056要求第二个字符为0x0

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 #返回值%eax和0x2比较

40102c:75 07 jne 401035 <phase\_4+0x29> #不相等则跳转至0x401035，引爆炸弹，说明需要输入两个数字

40102e:83 7c 24 08 0e cmpl $0xe,0x8(%rsp) #将(R[%rsp]+0x8)指向的值和0xe进行比较

401033:76 05 jbe 40103a <phase\_4+0x2e> #如果小于或等于0xe，则跳转至0x40103a，否则下一条指令引爆炸弹，说明第一个输入的数字要小于或等于0xe

401035:e8 00 04 00 00 callq 40143a <explode\_bomb>

40103a:ba 0e 00 00 00 mov $0xe,%edx #第3个参数，0xe

40103f:be 00 00 00 00 mov $0x0,%esi #第2个参数，0x0

401044:8b 7c 24 08 mov 0x8(%rsp),%edi #第1个参数，(R[%rsp]+0x8)指向的值，即输入的第一个数字

401048:e8 81 ff ff ff callq 400fce <func4>

40104d:85 c0 test %eax,%eax

40104f:75 07 jne 401058 <phase\_4+0x4c> #返回值不等于0,则跳转至0x401058，引爆炸弹，说明func4的返回值应该是0

401051:83 7c 24 0c 00 cmpl $0x0,0xc(%rsp) #第二个字符和0x0比较

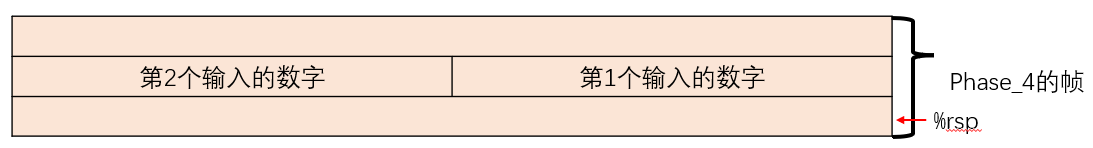
401056:74 05 je 40105d <phase\_4+0x51> #如果相等，则跳转至0x40105d，否则下一条指令引爆炸弹，说明第二个字符为0

401058:e8 dd 03 00 00 callq 40143a <explode\_bomb>

40105d:48 83 c4 18 add $0x18,%rsp

401061:c3 retq

callq 400bf0 <\_\_isoc99\_sscanf@plt>后phase\_4的帧如下：



**phase\_5**

**答案：**ascii码十六位数表示的低四位分别为"9"，"f"，"e"，"5"，"6"，"7"的任意字符组合，如"ionefg"，包括大小写字母之外的字符

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 目标字符 | f | l | y | e | r | s |
| 低四位 | 9 | f | e | 5 | 6 | 7 |

0000000000401062 <phase\_5>:

401062:53 push %rbx

401063:48 83 ec 20 sub $0x20,%rsp

401067:48 89 fb mov %rdi,%rbx #保存输入字符的首地址

40106a:64 48 8b 04 25 28 00 mov %fs:0x28,%rax #栈保护机制，在局部缓冲区与栈状态之间存储一个金丝雀值

401071:00 00

401073:48 89 44 24 18 mov %rax,0x18(%rsp) #金丝雀值存储在相对于%rsp偏移0x18的位置

401078:31 c0 xor %eax,%eax #清零

40107a:e8 9c 02 00 00 callq 40131b <string\_length>

40107f:83 f8 06 cmp $0x6,%eax #说明要输入字符的长度为0x6

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 #将相对输入字符首地址偏移(R[%rax])的字符保存在%ecx

40108f:88 0c 24 mov %cl,(%rsp) #将%ecx中的字符保存在%rsp指向的位置

401092:48 8b 14 24 mov (%rsp),%rdx #将%rsp指向的位置的字符保存在%rdx中

401096:83 e2 0f and $0xf,%edx #按位与，保留低4位，清零其他位

401099:0f b6 92 b0 24 40 00 movzbl 0x4024b0(%rdx),%edx #将首地址为0x4024b0，偏移为R[%rdx]的字符保存在%rdx中

4010a0:88 54 04 10 mov %dl,0x10(%rsp,%rax,1) #将%dl中的值保存在帧中，相对于%rsp的偏移值为0x10+R[%rax]

4010a4:48 83 c0 01 add $0x1,%rax #rax的值加1

4010a8:48 83 f8 06 cmp $0x6,%rax #R[%rax]与0x6比较

4010ac:75 dd jne 40108b <phase\_5+0x29> #不相等则跳转至0x40108b，循环遍历输入的字符

4010ae:c6 44 24 16 00 movb $0x0,0x16(%rsp) #将相对于%rsp的偏移为0x16的值设置为0,结果见图所示

4010b3:be 5e 24 40 00 mov $0x40245e,%esi #设置%esi的值为0x40245e

4010b8:48 8d 7c 24 10 lea 0x10(%rsp),%rdi #设置%rdi的值为(R[%rsp]+0x10)

4010bd:e8 76 02 00 00 callq 401338 <strings\_not\_equal> #判断地址0x40245e的字符和帧中(R[%rsp]+0x10)处的字符是否相等

4010c2:85 c0 test %eax,%eax

4010c4:74 13 je 4010d9 <phase\_5+0x77> #相等则跳转至0x4010d9

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 #%eax设置为0x0

4010d7:eb b2 jmp 40108b <phase\_5+0x29> #跳转至0x40108b

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

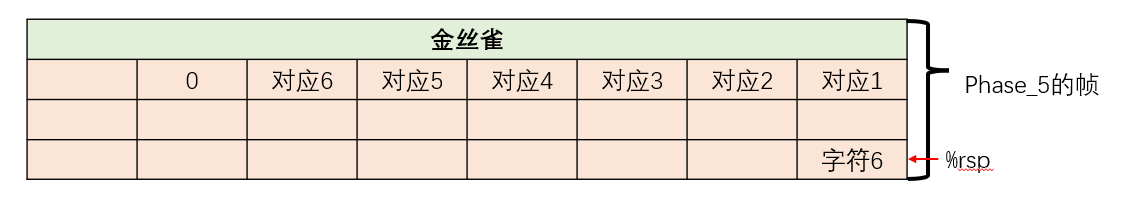
4010e7:74 05 je 4010ee <phase\_5+0x8c>

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



phase\_5需要输入6个字符，并将输入的字符的低4位做为偏移值，以0x4024b0为首地址，访问对应的字符，对应关系如下所示，并将其保存在帧中，对应的帧如图所示。

(gdb) print \* (char \*) 0x4024b0

$30 = 109 'm'

(gdb) print \* (char \*) 0x4024b1

$31 = 97 'a'

(gdb) print \* (char \*) 0x4024b2

$32 = 100 'd'

(gdb) print \* (char \*) 0x4024b3

$33 = 117 'u'

(gdb) print \* (char \*) 0x4024b4

$34 = 105 'i'

(gdb) print \* (char \*) 0x4024b5

$35 = 101 'e'

(gdb) print \* (char \*) 0x4024b6

$36 = 114 'r'

(gdb) print \* (char \*) 0x4024b7

$37 = 115 's'

(gdb) print \* (char \*) 0x4024b8

$38 = 110 'n'

(gdb) print \* (char \*) 0x4024b9

$39 = 102 'f'

(gdb) print \* (char \*) 0x4024ba

$40 = 111 'o'

(gdb) print \* (char \*) 0x4024bb

$41 = 116 't'

(gdb) print \* (char \*) 0x4024bc

$42 = 118 'v'

(gdb) print \* (char \*) 0x4024bd

$43 = 98 'b'

(gdb) print \* (char \*) 0x4024be

$44 = 121 'y'

(gdb) print \* (char \*) 0x4024bf

$45 = 108 'l'

目标字符的首地址为0x40245e，访问可知目标字符为flyers，需要输入字符的低4位分别为9，f，e，5，6，7

(gdb) print \* (char \*) 0x40245e

$63 = 102 'f'

(gdb) print \* (char \*) 0x40245f

$64 = 108 'l'

(gdb) print \* (char \*) 0x402460

$65 = 121 'y'

(gdb) print \* (char \*) 0x402461

$66 = 101 'e'

(gdb) print \* (char \*) 0x402462

$67 = 114 'r'

(gdb) print \* (char \*) 0x402463

$68 = 115 's'

**phase\_6**

**答案：**"4 3 2 1 6 5"

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 #把栈指针保存在%r13中

401103:48 89 e6 mov %rsp,%rsi #把栈指针保存在%rsi中

401106:e8 51 03 00 00 callq 40145c <read\_six\_numbers> #读取6个数字，并保存在帧上

40110b:49 89 e6 mov %rsp,%r14 #把栈指针保存在%r14中

40110e:41 bc 00 00 00 00 mov $0x0,%r12d #设置%r12d的值为0x0

401114:4c 89 ed mov %r13,%rbp #将R[%r13]保存在%rbp中

401117:41 8b 45 00 mov 0x0(%r13),%eax #将帧上相对于R[%r13]偏移为0x0的值保存在%eax中

40111b:83 e8 01 sub $0x1,%eax #%eax中的值减1

40111e:83 f8 05 cmp $0x5,%eax #%eax中的值和0x5比较

401121:76 05 jbe 401128 <phase\_6+0x34> #小于或等于0x5则跳转至0x401128，否则下一条指令引爆炸弹，说明输入的数字要小于或等于0x6

401123:e8 12 03 00 00 callq 40143a <explode\_bomb>

401128:41 83 c4 01 add $0x1,%r12d #%r12d的值加1，感觉%12有点像计数器

40112c:41 83 fc 06 cmp $0x6,%r12d #%r12d中的值和0x6比较

401130:74 21 je 401153 <phase\_6+0x5f> #如果相等，则跳转至0x401153，跳出外面的大循环

401132:44 89 e3 mov %r12d,%ebx #将R[%r12d]保存在%ebx中

401135:48 63 c3 movslq %ebx,%rax #将R[%ebx]保存在%rax中,因为%ebx是双字，%rax是四字，所以要用movslq

401138:8b 04 84 mov (%rsp,%rax,4),%eax #将帧上相对于栈指针偏移为(R[%rax]\*4)的值保存在%eax中

40113b:39 45 00 cmp %eax,0x0(%rbp) #%eax中的值和相对R[%rbp](栈指针)偏移0x0指向的值比较

40113e:75 05 jne 401145 <phase\_6+0x51> #如果不相等，则跳转至0x401145，说明第一个数和第二个数不能相等

401140:e8 f5 02 00 00 callq 40143a <explode\_bomb>

401145:83 c3 01 add $0x1,%ebx #%ebx中的值加1,感觉%ebx有点像计数器

401148:83 fb 05 cmp $0x5,%ebx #%eax中的值和0x5比较

40114b:7e e8 jle 401135 <phase\_6+0x41> #如果小于或等于0x5则跳转至0x401135，进入循环，遍历帧上保存的输入数字，要求所有数字不等于第一个数字

40114d:49 83 c5 04 add $0x4,%r13 #跳出循环时%ebx的值为0x6，%r13中的值加0x4,指向帧上保存的第二个输入数字

401151:eb c1 jmp 401114 <phase\_6+0x20> #跳转至0x401114，在上面的循环之外还有更大的循环，这个大的循环要求所有输入的数字小于或等于0x6，且后面的数不等于前面的数

401153:48 8d 74 24 18 lea 0x18(%rsp),%rsi #将帧上相对于栈指针偏移为0x18的地址保存在%rax

401158:4c 89 f0 mov %r14,%rax #将栈指针保存在%rax中

40115b:b9 07 00 00 00 mov $0x7,%ecx #将%ecx中的值设置为0x7

401160:89 ca mov %ecx,%edx #将R[%ecx]保存在%edx

401162:2b 10 sub (%rax),%edx #%edx的值减去R[%rax]指向的值

401164:89 10 mov %edx,(%rax) #将结果保存在R[%rax]指向的地址中

401166:48 83 c0 04 add $0x4,%rax #%rax的值加0x4

40116a:48 39 f0 cmp %rsi,%rax #比较%rsi和%rax的值

40116d:75 f1 jne 401160 <phase\_6+0x6c> #如果不相等，则跳转至0x401160，进入新的循环，使帧上的保存的输入值变为和0x7的差

40116f:be 00 00 00 00 mov $0x0,%esi #%esi的值设置为0x0

401174:eb 21 jmp 401197 <phase\_6+0xa3>

401176:48 8b 52 08 mov 0x8(%rdx),%rdx #将(R[%rdx]+0x8)指向的值保存在%rdx中

40117a:83 c0 01 add $0x1,%eax #%eax的值加1

40117d:39 c8 cmp %ecx,%eax #比较%ecx和%eax的值

40117f:75 f5 jne 401176 <phase\_6+0x82> #如果不相等，则跳转至0x401176，进入循环，目的是将%rdx的值变动(帧上相对于栈指针偏移为R[%rsi]的值-1)次

401181:eb 05 jmp 401188 <phase\_6+0x94>

401183:ba d0 32 60 00 mov $0x6032d0,%edx #%edx的值设置为0x6032d0

401188:48 89 54 74 20 mov %rdx,0x20(%rsp,%rsi,2) #将%rdx指向的值保存到帧中栈地址为0x20,偏移为0x20+2\*R[%rsi]

40118d:48 83 c6 04 add $0x4,%rsi #%rsi的值加0x4

401191:48 83 fe 18 cmp $0x18,%rsi #%rsi的值和0x18比较

401195:74 14 je 4011ab <phase\_6+0xb7> #相等则跳转至0x4011ab,此时的帧如图所示

401197:8b 0c 34 mov (%rsp,%rsi,1),%ecx #将帧上相对于栈指针偏移为R[%rsi]的值保存在%ecx

40119a:83 f9 01 cmp $0x1,%ecx #比较%ecx的值和0x1

40119d:7e e4 jle 401183 <phase\_6+0x8f> #如果小于或等于0x1则跳转至0x401183

40119f:b8 01 00 00 00 mov $0x1,%eax #如果大于0x1,将%eax设置为0x1

4011a4:ba d0 32 60 00 mov $0x6032d0,%edx #%edx的值设置为0x6032d0

4011a9:eb cb jmp 401176 <phase\_6+0x82> #跳转到0x401176

4011ab:48 8b 5c 24 20 mov 0x20(%rsp),%rbx #将帧上相对于栈指针偏移为0x20的值保存在%rbx中

4011b0:48 8d 44 24 28 lea 0x28(%rsp),%rax #将帧上相对于栈指针偏移为0x28的地址保存在%rax中

4011b5:48 8d 74 24 50 lea 0x50(%rsp),%rsi #将帧上相对于栈指针偏移为0x50的地址保存在%rsi

4011ba:48 89 d9 mov %rbx,%rcx #将R[%rbx]保存在%rcx中

4011bd:48 8b 10 mov (%rax),%rdx #将R[%rax]指向的值保存在%rdx中

4011c0:48 89 51 08 mov %rdx,0x8(%rcx) #将R[%rdx]保存在(R[%rcx]+0x8)指向的位置

4011c4:48 83 c0 08 add $0x8,%rax #%rax的值加0x8

4011c8:48 39 f0 cmp %rsi,%rax #比较%rsi和%rax的值

4011cb:74 05 je 4011d2 <phase\_6+0xde> #相等则跳转至0x4011d2

4011cd:48 89 d1 mov %rdx,%rcx

4011d0:eb eb jmp 4011bd <phase\_6+0xc9> #跳转到0x4011bd，遍历改变的值2至改变的值6,将后一个改变的值保存在地址为前一个改变的值+0x8的位置

4011d2:48 c7 42 08 00 00 00 movq $0x0,0x8(%rdx) #将0x0保存在(R[%rdx]+0x8)指向的位置,将地址为改变的值6+0x8的位置设为0x0

4011d9:00

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

0x401176-0x4011a9创建改变的值，如果(0x7-输入数字)小于1,则改变的值就是0x6032d0指向的值，否则改变的值就是0x6032d0以0x8(%rdx)改变的地址指向的值

0x4011ab-0x4011d9将后一个改变的值保存在地址为前一个改变的值+0x8的位置

0x4011d9后帧如图所示



输入数字与改变的值的对应关系如图所示



0x4011d9之后的值令遍历改变的值，将改变的值作为地址对应的值和改变的值+0x8作为地址对应的值进行比较，要求将改变的值作为地址对应的值更大。换句话说，让上一个改变的值与下一个改变的值进行对比，要求上一个改变的值更大。