CSAPP lab2 bomb（深入了解计算机系统 实验二）

原创

小帅比simon 于 2017-03-24 18:18:29 发布 11738 收藏 53

分类专栏： 操作系统

本文链接：https://blog.csdn.net/lzjsqn/article/details/65635265

这个问题还得用GDB调试来做。

截图做笔记吧，实在写不动了！

1.执行反汇编

```
obj-dump -D bomb > mysrc.S
```

得到可执行文件的机器级程序（汇编文件）。

2.搜索main（每一个应用程序都是从main函数开始）

找到如下内容：

```
0000000000400da0 <main>:
  400da0:   53                      push   %rbx
  400da1:   83 ff 01                cmp    $0x1,%edi
  400da4:   75 10                   jne    400db6 <main+0x16>
  400da6:   48 8b 05 9b 29 20 00    mov    0x20299b(%rip),%rax        # 603748 <stdin@@GLIBC_2.2.5>
  400dad:   48 89 05 b4 29 20 00    mov    %rax,0x2029b4(%rip)        # 603768 <infile>
  400db4:   eb 63                   jmp    400e19 <main+0x79>
  400db6:   48 89 f3                mov    %rsi,%rbx
  400db9:   83 ff 02                cmp    $0x2,%edi
  400dbc:   75 3a                   jne    400df8 <main+0x58>
  400dbe:   48 8b 7e 08             mov    0x8(%rsi),%rdi
  400dc2:   be b4 22 40 00          mov    $0x4022b4,%esi
  400dc7:   e8 44 fe ff ff          callq  400c10 <fopen@plt>
  400dcc:   48 89 05 95 29 20 00    mov    %rax,0x202995(%rip)        # 603768 <infile>
  400dd3:   48 85 c0                test   %rax,%rax
  400dd6:   75 41                   jne    400e19 <main+0x79>
  400dd8:   48 8b 4b 08             mov    0x8(%rbx),%rcx
  400ddc:   48 8b 13                mov    (%rbx),%rdx
  400ddf:   be b6 22 40 00          mov    $0x4022b6,%esi
  400de4:   bf 01 00 00 00          mov    $0x1,%edi
  400de9:   e8 12 fe ff ff          callq  400c00 <__printf_chk@plt>
  400dee:   bf 08 00 00 00          mov    $0x8,%edi
  400df3:   e8 28 fe ff ff          callq  400c20 <exit@plt>
  400df8:   48 8b 16                mov    (%rsi),%rdx
  400dfb:   be d3 22 40 00          mov    $0x4022d3,%esi
  400e00:   bf 01 00 00 00          mov    $0x1,%edi
  400e05:   b8 00 00 00 00          mov    $0x0,%eax
  400e0a:   e8 f1 fd ff ff          callq  400c00 <__printf_chk@plt>
  400e0f:   bf 08 00 00 00          mov    $0x8,%edi
  400e14:   e8 07 fe ff ff          callq  400c20 <exit@plt>
  400e19:   e8 84 05 00 00          callq  4013a2 <initialize_bomb>
  400e1e:   bf 38 23 40 00          mov    $0x402338,%edi
  400e23:   e8 e8 fc ff ff          callq  400b10 <puts@plt>
  400e28:   bf 78 23 40 00          mov    $0x402378,%edi
  400e2d:   e8 de fc ff ff          callq  400b10 <puts@plt>
  400e32:   e8 67 06 00 00          callq  40149e <read_line>
  400e37:   48 89 c7                mov    %rax,%rdi
  400e3a:   e8 a1 00 00 00          callq  400ee0 <phase_1>
  400e3f:   e8 80 07 00 00          callq  4015c4 <phase_defused>
  400e44:   bf a8 23 40 00          mov    $0x4023a8,%edi
  400e49:   e8 c2 fc ff ff          callq  400b10 <puts@plt>
  400e4e:   e8 4b 06 00 00          callq  40149e <read_line>
  400e53:   48 89 c7                mov    %rax,%rdi
  400e56:   e8 a1 00 00 00          callq  400efc <phase_2>
  400e5b:   e8 64 07 00 00          callq  4015c4 <phase_defused>
```

再从上述内容找到如下内容：

```
400e3a: e8 a1 00 00 00          callq  400ee0 <phase_1>
```

在这条语句之前的内容都是初始化相关的操作，从这里开始才是开始执行拆炸弹的任务，phase_1表明这是拆炸弹的第一关。可以看到一直到phase_6，说明有6个关卡，而且最后还有一个隐藏的关口！于是我们可以利用GDB一个关口一个关口的设置断点，看看每一个关口如何判断我们输入的数据是否是正确的拆弹密码！

3.另外在反汇编文件里再找到phase_1的汇编代码如下：

```
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
```

仔细思考，炸弹就是通过explode_bomb函数引爆，所有后面就好办了！只要避免程序调用该函数就可以拆除炸弹。

4.利用GDB启动bomb程序，并设置断点：

```
zhenjun@lzj:~/Downloads/bomb$ gdb bomb
GNU gdb (Ubuntu 7.11.1-0ubuntu1~16.04) 7.11.1
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...done.
(gdb) b phase_1
Breakpoint 1 at 0x400ee0
(gdb) b phase_2
Breakpoint 2 at 0x400efc
(gdb) b explode_bomb
Breakpoint 3 at 0x40143a
(gdb) run
Starting program: /home/zhenjun/Downloads/bomb/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
1234

Breakpoint 1, 0x0000000000400ee0 in phase_1 ()
(gdb) disas
Dump of assembler code for function phase_1:
=> 0x0000000000400ee0 <+0>:     sub    $0x8,%rsp
   0x0000000000400ee4 <+4>:     mov    $0x402400,%esi
   0x0000000000400ee9 <+9>:     callq  0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   %eax,%eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    callq  0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    $0x8,%rsp
   0x0000000000400efb <+27>:    retq
End of assembler dump.
(gdb) x/sb 0x402400
0x402400:       "Border relations with Canada have never been better."
(gdb)
```

因为程序会通过explode_bomb引爆炸弹，所以在phase_1和phase_2以及explode_bomb都设置断点，看看程序的具体执行流程，当开始run程序以后，会在phase_1停下来，之后利用disas反汇编可以查看phase_1的具体汇编代码，发现这个函数的流程就是：

（1）调整栈指针

（2）将0x402400赋值到%esi寄存器，根据X86-64的参数传递规则，我们知道%esi寄存器保存的是函数调用的第二个参数。第一个参数就是我们通过标准输入/指定的输入设备文件输入的数据，这里我们简单输入“1234”，存放在%rax指向的地址处

（3）根据strings_not_equal函数的返回结果进行判断密码是否正确，返回值存放在%rax中

（4）比较%rax指向的字符串和0x402400处存放的字符串是否相等，%rax不为0，表示不相等，就爆炸，相等就过了第一关！跳转到`add $0x8,%rsp`处，结束pahse_1函数的调用！

所以我们只要知道0x402400处存放的字符串是什么就可以了！

输入：

`x/sb 0x402400`

就可以查看预设定的第一关密码了！可以得到第一关的密码是：

`Border relations with Canada have never been better.`

5.按照同样的方法，一个阶段一个阶段设置断点，反汇编具体的阶段函数，判断函数如何对比我们输入的数据就可以得到答案了！

6.第二阶段需要输入的数字是六个，后一个数字是前一个的两倍，第一个数字是1，所以密码是：

`1 2 4 8 16 32`

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
acbds

Breakpoint 1, 0x0000000000400efc in phase_2 ()
(gdb) disas
Dump of assembler code for function phase_2:
=> 0x0000000000400efc <+0>:     push   %rbp
   0x0000000000400efd <+1>:     push   %rbx
   0x0000000000400efe <+2>:     sub    $0x28,%rsp
   0x0000000000400f02 <+6>:     mov    %rsp,%rsi
   0x0000000000400f05 <+9>:     callq  0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmpl   $0x1,(%rsp)
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    callq  0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    -0x4(%rbx),%eax
   0x0000000000400f1a <+30>:    add    %eax,%eax
   0x0000000000400f1c <+32>:    cmp    %eax,(%rbx)
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    callq  0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    $0x4,%rbx
   0x0000000000400f29 <+45>:    cmp    %rbp,%rbx
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    0x4(%rsp),%rbx
   0x0000000000400f35 <+57>:    lea    0x18(%rsp),%rbp
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    $0x28,%rsp
   0x0000000000400f40 <+68>:    pop    %rbx
   0x0000000000400f41 <+69>:    pop    %rbp
   0x0000000000400f42 <+70>:    retq
End of assembler dump.
(gdb)
```


7.第三关反汇编以后，是一个含有switch语句的函数，要求输入两个数字，根据第一个数字决定分支语句，然后根据跳转表，就可以跳转到相应的语句块，判断第二个数字是否匹配，所以有多个密码组合。

```
(gdb) disas phase_3
Dump of assembler code for function phase_3:
   0x0000000000400f43 <+0>:     sub    $0x18,%rsp
   0x0000000000400f47 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000400f4c <+9>:     lea    0x8(%rsp),%rdx
   0x0000000000400f51 <+14>:    mov    $0x4025cf,%esi
   0x0000000000400f56 <+19>:    mov    $0x0,%eax
   0x0000000000400f5b <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000400f60 <+29>:    cmp    $0x1,%eax
   0x0000000000400f63 <+32>:    jg     0x400f6a <phase_3+39>
   0x0000000000400f65 <+34>:    callq  0x40143a <explode_bomb>
   0x0000000000400f6a <+39>:    cmpl   $0x7,0x8(%rsp)
   0x0000000000400f6f <+44>:    ja     0x400fad <phase_3+106>
   0x0000000000400f71 <+46>:    mov    0x8(%rsp),%eax
   0x0000000000400f75 <+50>:    jmpq   *0x402470(,%rax,8)
   0x0000000000400f7c <+57>:    mov    $0xcf,%eax
   0x0000000000400f81 <+62>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f83 <+64>:    mov    $0x2c3,%eax
   0x0000000000400f88 <+69>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f8a <+71>:    mov    $0x100,%eax
   0x0000000000400f8f <+76>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f91 <+78>:    mov    $0x185,%eax
   0x0000000000400f96 <+83>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f98 <+85>:    mov    $0xce,%eax
   0x0000000000400f9d <+90>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f9f <+92>:    mov    $0x2aa,%eax
   0x0000000000400fa4 <+97>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400fa6 <+99>:    mov    $0x147,%eax
   0x0000000000400fab <+104>:   jmp    0x400fbe <phase_3+123>
   0x0000000000400fad <+106>:   callq  0x40143a <explode_bomb>
   0x0000000000400fb2 <+111>:   mov    $0x0,%eax
   0x0000000000400fb7 <+116>:   jmp    0x400fbe <phase_3+123>
   0x0000000000400fb9 <+118>:   mov    $0x137,%eax
   0x0000000000400fbe <+123>:   cmp    0xc(%rsp),%eax
   0x0000000000400fc2 <+127>:   je     0x400fc9 <phase_3+134>
   0x0000000000400fc4 <+129>:   callq  0x40143a <explode_bomb>
   0x0000000000400fc9 <+134>:   add    $0x18,%rsp
   0x0000000000400fcd <+138>:   retq
End of assembler dump.
```


好比

```
    switch(n)
    {
        case n1:
            if(m != m1)
                explode_bomb();
            break;
    ...   ...  ...
        case n7:
            if(m != m7)
                explode_bomb();
            break;
        default:
            explode_bomb();
            break;
    }
```

输入的第一个数就是n，第二个数就是m。从汇编代码可以看出，要求n小于等于7，所以又0-7八种可能，分别是：

n	m
0	0xcf
1	0x137

n	m
2	0x2c3
3	0x100
4	0x185
5	0xce
6	0x2aa
7	0x147

这里我们选择1和311输入（0x137的十进制），注意要输入为十进制。

8.第四关主要是一个递归函数，主要是找到递归的终止条件，

```
Starting program: /home/zhenjun/Downloads/bomb/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
1 311
Halfway there!
h jk

Breakpoint 1, 0x000000000040100c in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x000000000040100c <+0>:     sub    $0x18,%rsp
   0x0000000000401010 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000401015 <+9>:     lea    0x8(%rsp),%rdx
   0x000000000040101a <+14>:    mov    $0x4025cf,%esi
   0x000000000040101f <+19>:    mov    $0x0,%eax
   0x0000000000401024 <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000401029 <+29>:    cmp    $0x2,%eax
   0x000000000040102c <+32>:    jne    0x401035 <phase_4+41>
   0x000000000040102e <+34>:    cmpl   $0xe,0x8(%rsp)
   0x0000000000401033 <+39>:    jbe    0x40103a <phase_4+46>
   0x0000000000401035 <+41>:    callq  0x40143a <explode_bomb>
   0x000000000040103a <+46>:    mov    $0xe,%edx
   0x000000000040103f <+51>:    mov    $0x0,%esi
   0x0000000000401044 <+56>:    mov    0x8(%rsp),%edi
   0x0000000000401048 <+60>:    callq  0x400fce <func4>
   0x000000000040104d <+65>:    test   %eax,%eax
   0x000000000040104f <+67>:    jne    0x401058 <phase_4+76>
   0x0000000000401051 <+69>:    cmpl   $0x0,0xc(%rsp)
   0x0000000000401056 <+74>:    je     0x40105d <phase_4+81>
   0x0000000000401058 <+76>:    callq  0x40143a <explode_bomb>
   0x000000000040105d <+81>:    add    $0x18,%rsp
   0x0000000000401061 <+85>:    retq
End of assembler dump.
(gdb)
```

这里也是要求输入两个数字，最后一个一定是0，第一个是0xe的因子，实质上只要是func4函数能够返回0的的输入值都可以。因为我们输入的第一个数回作为func4函数的第一个参数！他的第二个参数是0，第三个参数是0xe。然后反汇编func4函数就可以判断当输入的第一个数满足什么条件的时候函数才会返回0，如果不返回0，就会爆炸！

```
(gdb) disas func4
Dump of assembler code for function func4:
   0x0000000000400fce <+0>:     sub    $0x8,%rsp
   0x0000000000400fd2 <+4>:     mov    %edx,%eax
   0x0000000000400fd4 <+6>:     sub    %esi,%eax
   0x0000000000400fd6 <+8>:     mov    %eax,%ecx
   0x0000000000400fd8 <+10>:    shr    $0x1f,%ecx
   0x0000000000400fdb <+13>:    add    %ecx,%eax
   0x0000000000400fdd <+15>:    sar    %eax
   0x0000000000400fdf <+17>:    lea    (%rax,%rsi,1),%ecx
   0x0000000000400fe2 <+20>:    cmp    %edi,%ecx
   0x0000000000400fe4 <+22>:    jle    0x400ff2 <func4+36>
   0x0000000000400fe6 <+24>:    lea    -0x1(%rcx),%edx
   0x0000000000400fe9 <+27>:    callq  0x400fce <func4>
   0x0000000000400fee <+32>:    add    %eax,%eax
   0x0000000000400ff0 <+34>:    jmp    0x401007 <func4+57>
   0x0000000000400ff2 <+36>:    mov    $0x0,%eax
   0x0000000000400ff7 <+41>:    cmp    %edi,%ecx
   0x0000000000400ff9 <+43>:    jge    0x401007 <func4+57>
   0x0000000000400ffb <+45>:    lea    0x1(%rcx),%esi
   0x0000000000400ffe <+48>:    callq  0x400fce <func4>
   0x0000000000401003 <+53>:    lea    0x1(%rax,%rax,1),%eax
   0x0000000000401007 <+57>:    add    $0x8,%rsp
   0x000000000040100b <+61>:    retq
End of assembler dump.
(gdb)                                   http://blog.csdn.net/lzjsqn
```

这里我们的密码是：

```
7 0
```

也许还有其他答案，但没有往下分析，因为输入第一个数是7的时候，明显满足func4返回0，能完成拆弹任务！

9.第五关的反汇编代码是：

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x0000000000401062 <+0>:     push   %rbx
   0x0000000000401063 <+1>:     sub    $0x20,%rsp
   0x0000000000401067 <+5>:     mov    %rdi,%rbx
   0x000000000040106a <+8>:     mov    %fs:0x28,%rax
   0x0000000000401073 <+17>:    mov    %rax,0x18(%rsp)
   0x0000000000401078 <+22>:    xor    %eax,%eax
   0x000000000040107a <+24>:    callq  0x40131b <string_length>
   0x000000000040107f <+29>:    cmp    $0x6,%eax
   0x0000000000401082 <+32>:    je     0x4010d2 <phase_5+112>
   0x0000000000401084 <+34>:    callq  0x40143a <explode_bomb>
   0x0000000000401089 <+39>:    jmp    0x4010d2 <phase_5+112>
   0x000000000040108b <+41>:    movzbl (%rbx,%rax,1),%ecx
   0x000000000040108f <+45>:    mov    %cl,(%rsp)
   0x0000000000401092 <+48>:    mov    (%rsp),%rdx
   0x0000000000401096 <+52>:    and    $0xf,%edx
   0x0000000000401099 <+55>:    movzbl 0x4024b0(%rdx),%edx
   0x00000000004010a0 <+62>:    mov    %dl,0x10(%rsp,%rax,1)
   0x00000000004010a4 <+66>:    add    $0x1,%rax
   0x00000000004010a8 <+70>:    cmp    $0x6,%rax
   0x00000000004010ac <+74>:    jne    0x40108b <phase_5+41>
   0x00000000004010ae <+76>:    movb   $0x0,0x16(%rsp)
   0x00000000004010b3 <+81>:    mov    $0x40245e,%esi
   0x00000000004010b8 <+86>:    lea    0x10(%rsp),%rdi
   0x00000000004010bd <+91>:    callq  0x401338 <strings_not_equal>
   0x00000000004010c2 <+96>:    test   %eax,%eax
   0x00000000004010c4 <+98>:    je     0x4010d9 <phase_5+119>
   0x00000000004010c6 <+100>:   callq  0x40143a <explode_bomb>
   0x00000000004010cb <+105>:   nopl   0x0(%rax,%rax,1)
   0x00000000004010d0 <+110>:   jmp    0x4010d9 <phase_5+119>
   0x00000000004010d2 <+112>:   mov    $0x0,%eax
   0x00000000004010d7 <+117>:   jmp    0x40108b <phase_5+41>
   0x00000000004010d9 <+119>:   mov    0x18(%rsp),%rax
   0x00000000004010de <+124>:   xor    %fs:0x28,%rax
   0x00000000004010e7 <+133>:   je     0x4010ee <phase_5+140>
   0x00000000004010e9 <+135>:   callq  0x400b30 <__stack_chk_fail@plt>
   0x00000000004010ee <+140>:   add    $0x20,%rsp
   0x00000000004010f2 <+144>:   pop    %rbx
   0x00000000004010f3 <+145>:   retq
End of assembler dump.
(gdb) x/sb 0x40245e
0x40245e:       "flyers"
(gdb) x/sb 0x4024b0
0x4024b0 <array.3449>:  "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
(gdb)
```


大体意思是，要求输入六个字符，然后根据每一个字符的低四位（ASCII码是8位），作为索引/数组下标，这里的数组是0x4024b0开头一个字符串：

`maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?`

根据得到的索引从该字符串中取出六个字符，这些字符必须和0x40245e处存放的`flyers`相等才可以通关。所以我们从0x4024b0处的字符串分别找到`flyers`里面每个字符的下标，这些下标是我们输入的字符的低四位，然后对照ASCII表，既可找到我们应该输入的字符串！
比如f在0x4024b0处的字符串中是第9个，对应的二进制就是1001，也就是我们输入的第一个字符的低四位是1001，对照ASCII表可以得到对应的是`i`或者'I'（也就是大小写字母都满足条件），

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

以此类推密码是：
`ionefg`

10.第六关的反汇编很长，截取部分如下：

```
Dump of assembler code for function phase_6:
   0x00000000004010f4 <+0>:     push   %r14
   0x00000000004010f6 <+2>:     push   %r13
   0x00000000004010f8 <+4>:     push   %r12
   0x00000000004010fa <+6>:     push   %rbp
   0x00000000004010fb <+7>:     push   %rbx
   0x00000000004010fc <+8>:     sub    $0x50,%rsp
   0x0000000000401100 <+12>:    mov    %rsp,%r13
   0x0000000000401103 <+15>:    mov    %rsp,%rsi
   0x0000000000401106 <+18>:    callq  0x40145c <read_six_numbers>
   0x000000000040110b <+23>:    mov    %rsp,%r14
   0x000000000040110e <+26>:    mov    $0x0,%r12d
   0x0000000000401114 <+32>:    mov    %r13,%rbp
   0x0000000000401117 <+35>:    mov    0x0(%r13),%eax
   0x000000000040111b <+39>:    sub    $0x1,%eax
   0x000000000040111e <+42>:    cmp    $0x5,%eax
   0x0000000000401121 <+45>:    jbe    0x401128 <phase_6+52>
   0x0000000000401123 <+47>:    callq  0x40143a <explode_bomb>
   0x0000000000401128 <+52>:    add    $0x1,%r12d
   0x000000000040112c <+56>:    cmp    $0x6,%r12d
   0x0000000000401130 <+60>:    je     0x401153 <phase_6+95>
   0x0000000000401132 <+62>:    mov    %r12d,%ebx
   0x0000000000401135 <+65>:    movslq %ebx,%rax
   0x0000000000401138 <+68>:    mov    (%rsp,%rax,4),%eax
   0x000000000040113b <+71>:    cmp    %eax,0x0(%rbp)
   0x000000000040113e <+74>:    jne    0x401145 <phase_6+81>
   0x0000000000401140 <+76>:    callq  0x40143a <explode_bomb>
   0x0000000000401145 <+81>:    add    $0x1,%ebx
   0x0000000000401148 <+84>:    cmp    $0x5,%ebx
   0x000000000040114b <+87>:    jle    0x401135 <phase_6+65>
   0x000000000040114d <+89>:    add    $0x4,%r13
   0x0000000000401151 <+93>:    jmp    0x401114 <phase_6+32>
   0x0000000000401153 <+95>:    lea    0x18(%rsp),%rsi
   0x0000000000401158 <+100>:   mov    %r14,%rax
   0x000000000040115b <+103>:   mov    $0x7,%ecx
   0x0000000000401160 <+108>:   mov    %ecx,%edx
   0x0000000000401162 <+110>:   sub    (%rax),%edx
   0x0000000000401164 <+112>:   mov    %edx,(%rax)
   0x0000000000401166 <+114>:   add    $0x4,%rax
   0x000000000040116a <+118>:   cmp    %rsi,%rax
   0x000000000040116d <+121>:   jne    0x401160 <phase_6+108>
   0x000000000040116f <+123>:   mov    $0x0,%esi
   0x0000000000401174 <+128>:   jmp    0x401197 <phase_6+163>
   0x0000000000401176 <+130>:   mov    0x8(%rdx),%rdx
   0x000000000040117a <+134>:   add    $0x1,%eax
   0x000000000040117d <+137>:   cmp    %ecx,%eax
   0x000000000040117f <+139>:   jne    0x401176 <phase_6+130>
   0x0000000000401181 <+141>:   jmp    0x401188 <phase_6+148>
---Type <return> to continue, or q <return> to quit---
```


从代码可以知道是要输入6个数字，看做一个数组a[6],然后对于每一个数组元素，在栈里面存放为7-a[i]，其中i属于0–5，查看0x6032d0处的内容，可以下发现：

```
End of assembler dump.
(gdb) x/24 0x6032d0
0x6032d0 <node1>:       332     1       6304480 0
0x6032e0 <node2>:       168     2       6304496 0
0x6032f0 <node3>:       924     3       6304512 0
0x603300 <node4>:       691     4       6304528 0
0x603310 <node5>:       477     5       6304544 0
0x603320 <node6>:       443     6       0       0
(gdb)
```


发现其实是一个结构体，类似于

```
struct {
    int value;
    int order;
    node* next;
} node;
```

我们要做的，就是得到正确的 order（从大到小），保证栈里面的order对应的数据项依次从大到小即可。
比如说数据项为924的这个节点是这个结构体数组中最大的，他的order就应该是栈里面的第一个，他的order是3，所以我们栈里面存放的第一个数字就是3，就可以发现栈里面直观的顺序是：

`3 4 5 6 1 2`

对应的输入就应该是

`4 3 2 1 6 5`

拆弹任务完成！！！！

```
zhenjun@lzj:~/Downloads/bomb$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
1 311
Halfway there!
7 0
So you got that one.  Try this one.
ionefg
Good work!  On to the next...
4 3 2 1 6 5
Congratulations! You've defused the bomb!
zhenjun@lzj:~/Downloads/bomb$
```


至于隐藏任务以后有时间再说，现在感觉找不到工作了，好忧桑。。。。。。

另外附上两个帖子有助于帮助：不能照抄答案，每年的实验内容都会变化！！！！

http://wdxtub.com/2016/04/16/thick-csapp-lab-2/

http://www.cnblogs.com/chkkch/archive/2011/05/21/2052708.html