CMU 计算机课程 Bomb Lab 拆除过程解析

Yungang Bao(baoyungang@gmail.com) 2011/10/15

CMU 的计算机系统课程 Lab 有一个是拆炸弹：给一个二进制“炸弹”可执行文件，要猜对 6 条输入才不会引爆，既有挑战又有趣味。感兴趣的朋友可以尝试一下.

- **CMU 课程网址：** http://csapp.cs.cmu.edu/public/labs.html
- **炸弹下载地址：** http://csapp.cs.cmu.edu/public/bomb.tar

1. 结果

```
[ybao@a4 ~]$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
1 2 6 24 120 720
That's number 2.  Keep going!
1 b 214
Halfway there!
9
So you got that one.  Try this one.
opukma
Good work!  On to the next...
4 2 6 3 1 5
Curses, you've found the secret phase!
But finding it and solving it are quite different...
1001
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

2. 过程

这里主要使用 gdb 来拆炸弹。当然，用其他工具来辅助，应该可以更高效地完成。

(gdb) echo ========== Defuse Phase_1 ==========\n\n
========== Defuse Phase_1 ==========

(gdb) disassemble phase_1
Dump of assembler code for function phase_1:
0x08048b20 <phase_1+0>: push %ebp
…
0x08048b2c <phase_1+12>: push $0x80497c0
0x08048b31 <phase_1+17>: push %eax
0x08048b32 <phase_1+18>: call 0x8049030 <strings_not_equal>
…
0x08048b45 <phase_1+37>: pop %ebp
0x08048b46 <phase_1+38>: ret
End of assembler dump.
(gdb) x /32c 0x80497c0
0x80497c0: 80 'P' 117 'u' 98 'b' 108 'l' 105 'i' 99 'c' 32 ' ' 115 's'
0x80497c8: 112 'p' 101 'e' 97 'a' 107 'k' 105 'i' 110 'n' 103 'g' 32 ' '
0x80497d0: 105 'i' 115 's' 32 ' ' 118 'v' 101 'e' 114 'r' 121 'y' 32 ' '
0x80497d8: 101 'e' 97 'a' 115 's' 121 'y' 46 '.' 0 '\0' 37 '%' 100 'd'

调用 strings_not_equal()比较输入字符串与 0x80497c0 指向的字符串

0x80497c0 指向的字符串为”Public speaking is very easy.”

(gdb) echo ========== Defuse Phase_2 ==========\n\n
========== Defuse Phase_2 ==========

(gdb) disassemble phase_2
Dump of assembler code for function phase_2:
0x08048b48 <phase_2+0>: push %ebp
…
0x08048b56 <phase_2+14>: lea -0x18(%ebp),%eax
0x08048b59 <phase_2+17>: push %eax
0x08048b5a <phase_2+18>: push %edx
0x08048b5b <phase_2+19>: call 0x8048fd8 <read_six_numbers>
0x08048b60 <phase_2+24>: add $0x10,%esp
0x08048b63 <phase_2+27>: cmpl $0x1,-0x18(%ebp)
0x08048b67 <phase_2+31>: je 0x8048b6e <phase_2+38>
0x08048b69 <phase_2+33>: call 0x80494fc <explode_bomb>
0x08048b6e <phase_2+38>: mov $0x1,%ebx
0x08048b73 <phase_2+43>: lea -0x18(%ebp),%esi
0x08048b76 <phase_2+46>: lea 0x1(%ebx),%eax
0x08048b79 <phase_2+49>: imul -0x4(%esi,%ebx,4),%eax
0x08048b7e <phase_2+54>: cmp %eax,(%esi,%ebx,4)
0x08048b81 <phase_2+57>: je 0x8048b88 <phase_2+64>

调用 read_six_number 来输入 6 个数字

第一个数为”1”

对应代码：
For(i=1; i <=5; i++)
A[i]=A[i-1]*(i+1)

得到答案：
1 2 6 24 120 720

```
0x08048b83 <phase_2+59>:  call   0x80494fc <explode_bomb>
0x08048b88 <phase_2+64>:  inc    %ebx
0x08048b89 <phase_2+65>:  cmp    $0x5,%ebx
0x08048b8c <phase_2+68>:  jle    0x8048b76 <phase_2+46>
0x08048b8e <phase_2+70>:  lea    -0x28(%ebp),%esp
0x08048b91 <phase_2+73>:  pop    %ebx
0x08048b92 <phase_2+74>:  pop    %esi
0x08048b93 <phase_2+75>:  mov    %ebp,%esp
0x08048b95 <phase_2+77>:  pop    %ebp
0x08048b96 <phase_2+78>:  ret
End of assembler dump.
(gdb) disassemble read_six_numbers
Dump of assembler code for function read_six_numbers:
0x08048fd8 <read_six_numbers+0>:    push   %ebp
…
0x08048ff8 <read_six_numbers+32>:   push   %edx
0x08048ff9 <read_six_numbers+33>:   push   $0x8049b1b
0x08048ffe <read_six_numbers+38>:   push   %ecx
0x08048fff <read_six_numbers+39>:   call   0x8048860 <sscanf@plt>
0x08049004 <read_six_numbers+44>:   add    $0x20,%esp
0x08049007 <read_six_numbers+47>:   cmp    $0x5,%eax
0x0804900a <read_six_numbers+50>:   jg     0x8049011 <read_six_numbers+57>
0x0804900c <read_six_numbers+52>:   call   0x80494fc <explode_bomb>
0x08049011 <read_six_numbers+57>:   mov    %ebp,%esp
0x08049013 <read_six_numbers+59>:   pop    %ebp
0x08049014 <read_six_numbers+60>:   ret
End of assembler dump.
(gdb) x /32c 0x8049b1b
0x8049b1b:    37 '%'    100 'd'    32 ' '37 '%'    100 'd'    32 ' '37 '%'    100 'd'
0x8049b23:    32 ' '37 '%'    100 'd'    32 ' '37 '%'    100 'd'    32 ' '37 '%'
0x8049b2b:    100 'd'    0 '\0'66 'B'    97 'a'    100 'd'    32 ' '104 'h'    111 'o'
0x8049b33:    115 's'    116 't'    32 ' '40 '('49 '1'    41 ')'46 '.'10 '\n'
(gdb) echo ======================== Defuse Phase_3 ==============================\n\n
======================== Defuse Phase_3 ==============================

(gdb) disassemble phase_3
Dump of assembler code for function phase_3:
0x08048b98 <phase_3+0>:    push   %ebp
…
0x08048bad <phase_3+21>:  lea    -0xc(%ebp),%eax
0x08048bb0 <phase_3+24>:  push   %eax
0x08048bb1 <phase_3+25>:  push   $0x80497de
0x08048bb6 <phase_3+30>:  push   %edx
0x08048bb7 <phase_3+31>:  call   0x8048860 <sscanf@plt>
```

调用 sscanf 从字符串将数字按照 0x8049b1b 执行的格式解析出来

0x8049b1b 指向的字符串为”%d %d %d %d %d %d”

调用 sscanf 从字符串将数字按照 0x80497be 执行的格式解析: “%d %c %d”

…

0x08048bcd <phase_3+53>: ja 0x8048c88 <phase_3+240>

0x08048bd3 <phase_3+59>: mov -0xc(%ebp),%eax

0x08048bd6 <phase_3+62>: jmp *0x80497e8(,%eax,4)

0x08048bdd <phase_3+69>: lea 0x0(%esi),%esi

0x08048be0 <phase_3+72>: mov $0x71,%bl

0x08048be2 <phase_3+74>: cmpl $0x309,-0x4(%ebp)

0x08048be9 <phase_3+81>: je 0x8048c8f <phase_3+247>

0x08048bef <phase_3+87>: call 0x80494fc <explode_bomb>

0x08048bf4 <phase_3+92>: jmp 0x8048c8f <phase_3+247>

0x08048bf9 <phase_3+97>: lea 0x0(%esi,%eiz,1),%esi

0x08048c00 <phase_3+104>:mov $0x62,%bl

0x08048c02 <phase_3+106>:cmpl $0xd6,-0x4(%ebp)

0x08048c09 <phase_3+113>:je 0x8048c8f <phase_3+247>

0x08048c0f <phase_3+119>: call 0x80494fc <explode_bomb>

0x08048c14 <phase_3+124>:jmp 0x8048c8f <phase_3+247>

0x08048c16 <phase_3+126>:mov $0x62,%bl

0x08048c18 <phase_3+128>:cmpl $0x2f3,-0x4(%ebp)

0x08048c1f <phase_3+135>: je 0x8048c8f <phase_3+247>

0x08048c21 <phase_3+137>:call 0x80494fc <explode_bomb>

0x08048c26 <phase_3+142>:jmp 0x8048c8f <phase_3+247>

…

0x08048c8f <phase_3+247>: cmp -0x5(%ebp),%bl

0x08048c92 <phase_3+250>:je 0x8048c99 <phase_3+257>

0x08048c94 <phase_3+252>:call 0x80494fc <explode_bomb>

0x08048c99 <phase_3+257>:mov -0x18(%ebp),%ebx

0x08048c9c <phase_3+260>: mov %ebp,%esp

0x08048c9e <phase_3+262>: pop %ebp

0x08048c9f <phase_3+263>: ret

End of assembler dump.

(gdb) x /16c 0x80497de

0x80497de: 37 '%' 100 'd' 32 ' ' 37 '%' 99 'c' 32 ' ' 37 '%' 100 'd'

0x80497e6: 0 '\0' 0 '\0' -32 ' ' -117 '\213' 4 '\004' 8 '\b' 0 '\0' -116 '\214'

(gdb) x /32x 0x80497e8

0x80497e8: 0xe0 0x8b 0x04 0x08 0x00 0x8c 0x04 0x08

0x80497f0: 0x16 0x8c 0x04 0x08 0x28 0x8c 0x04 0x08

0x80497f8: 0x40 0x8c 0x04 0x08 0x52 0x8c 0x04 0x08

0x8049800: 0x64 0x8c 0x04 0x08 0x76 0x8c 0x04 0x08

(gdb) echo ========== Defuse Phase_4 ==========\n\n

========== Defuse Phase_4 ==========

(gdb) disassemble phase_4

Dump of assembler code for function phase_4:

0x08048ce0 <phase_4+0>: push %ebp

跳转向*(0x80497e8 + %eax*4),其中%eax 为输入的第一个数。见下页

对于第二个输入字符和第三个数字比较，取决于第一个数。这里我们选择第一个数为“1”,那么对应的输入为”1 b 214”

也可以选择偏移为 2，那么对应的输入为”2 b 755”

0x80497be 执行的格式解析:“%d %c %d”

偏移为 1 对应的跳转地址为 0x08048c00

…

```
0x08048cef <phase_4+15>:   push     %eax
0x08048cf0 <phase_4+16>:   push     $0x8049808
0x08048cf5 <phase_4+21>:   push     %edx
0x08048cf6 <phase_4+22>:   call     0x8048860 <sscanf@plt>
0x08048cfb <phase_4+27>:   add      $0x10,%esp
…
0x08048d14 <phase_4+52>:   push     %eax
0x08048d15 <phase_4+53>:   call     0x8048ca0 <func4>
0x08048d1a <phase_4+58>:   add      $0x10,%esp
0x08048d1d <phase_4+61>:   cmp      $0x37,%eax
0x08048d20 <phase_4+64>:   je       0x8048d27 <phase_4+71>
0x08048d22 <phase_4+66>:   call     0x80494fc <explode_bomb>
0x08048d27 <phase_4+71>:   mov      %ebp,%esp
0x08048d29 <phase_4+73>:   pop      %ebp
0x08048d2a <phase_4+74>:   ret
End of assembler dump.
(gdb) x /8c 0x8049808
0x8049808:    37 '%'     100 'd'     0 '\0' 103 'g'     105 'i'     97 'a'      110 'n'     116 't'
(gdb) disassemble func4
Dump of assembler code for function func4:
0x08048ca0 <func4+0>: push     %ebp
…
0x08048cb3 <func4+19>:     lea      -0x1(%ebx),%eax
0x08048cb6 <func4+22>:     push     %eax
0x08048cb7 <func4+23>:     call     0x8048ca0 <func4>
0x08048cbc <func4+28>:     mov      %eax,%esi
0x08048cbe <func4+30>:     add      $0xfffffff4,%esp
0x08048cc1 <func4+33>:     lea      -0x2(%ebx),%eax
0x08048cc4 <func4+36>:     push     %eax
0x08048cc5 <func4+37>:     call     0x8048ca0 <func4>
0x08048cca <func4+42>:     add      %esi,%eax
0x08048ccc <func4+44>:     jmp      0x8048cd5 <func4+53>
0x08048cce <func4+46>:     mov      %esi,%esi
0x08048cd0 <func4+48>:     mov      $0x1,%eax
…
0x08048cdd <func4+61>:     ret
End of assembler dump.
(gdb) echo ========================= Defuse Phase_5 ============================\n\n
========================= Defuse Phase_5 ============================

(gdb) disassemble phase_5
Dump of assembler code for function phase_5:
0x08048d2c <phase_5+0>:    push     %ebp
```

0x8049808 对应的格式为“%d “

将输入的数传递给 func4()

Func(i)的输出应该是 55，因此对应的 i=9

Func4(i)为递归函数：

```
Func4(i)
{
    if(i <=1)
        return 1;
    else return f(i-1) + f(i-2);
}
```

```
。 。 。
0x08048d3a <phase_5+14>:  push   %ebx
0x08048d3b <phase_5+15>:  call   0x8049018 <string_length>
0x08048d40 <phase_5+20>:  add    $0x10,%esp
0x08048d43 <phase_5+23>:  cmp    $0x6,%eax
0x08048d46 <phase_5+26>:  je     0x8048d4d <phase_5+33>
0x08048d48 <phase_5+28>:  call   0x80494fc <explode_bomb>
0x08048d4d <phase_5+33>:  xor    %edx,%edx
0x08048d4f <phase_5+35>:  lea    -0x8(%ebp),%ecx
0x08048d52 <phase_5+38>:  mov    $0x804b220,%esi
0x08048d57 <phase_5+43>:  mov    (%edx,%ebx,1),%al
0x08048d5a <phase_5+46>:  and    $0xf,%al
0x08048d5c <phase_5+48>:  movsbl %al,%eax
0x08048d5f <phase_5+51>:  mov    (%eax,%esi,1),%al
0x08048d62 <phase_5+54>:  mov    %al,(%edx,%ecx,1)
0x08048d65 <phase_5+57>:  inc    %edx
0x08048d66 <phase_5+58>:  cmp    $0x5,%edx
0x08048d69 <phase_5+61>:  jle    0x8048d57 <phase_5+43>
0x08048d6b <phase_5+63>:  movb   $0x0,-0x2(%ebp)
0x08048d6f <phase_5+67>:  add    $0xfffffff8,%esp
0x08048d72 <phase_5+70>:  push   $0x804980b
0x08048d77 <phase_5+75>:  lea    -0x8(%ebp),%eax
0x08048d7a <phase_5+78>:  push   %eax
0x08048d7b <phase_5+79>:  call   0x8049030 <strings_not_equal>
0x08048d80 <phase_5+84>:  add    $0x10,%esp
0x08048d83 <phase_5+87>:  test   %eax,%eax
0x08048d85 <phase_5+89>:  je     0x8048d8c <phase_5+96>
0x08048d87 <phase_5+91>:  call   0x80494fc <explode_bomb>
0x08048d8c <phase_5+96>:  lea    -0x18(%ebp),%esp
0x08048d8f <phase_5+99>:  pop    %ebx
0x08048d90 <phase_5+100>:pop    %esi
0x08048d91 <phase_5+101>:mov    %ebp,%esp
0x08048d93 <phase_5+103>:pop    %ebp
0x08048d94 <phase_5+104>:ret
---Type <return> to continue, or q <return> to quit---
End of assembler dump.
(gdb) x /32x 0x804b220
0x804b220 <array.123>:105 'i'    115 's'    114 'r'    118 'v'    101 'e'    97 'a'     119 'w'    104 'h'
0x804b228 <array.123+8>:  111 'o'    98 'b'     112 'p'    110 'n'    117 'u'    116 't'    102 'f'    103 'g'
 (gdb) x /32c 0x804980b
0x804980b:    103 'g'    105 'i'    97 'a'     110 'n'    116 't'    115 's'    0 '\0'0 '\0'
```

输入长度为 6 的字符串

以输入字符串（in）为索引，在 0x0x804b220（str）对应的字符串中找出新的字符串(new)

```
For(i=0; i<=5; i++)
   new[i]=str[in[i]&0xf
```

将得到的新字符串 new 与 0x804980b(“giants”)存储的字符串比较

(gdb) echo ========== Defuse Phase_6 ==========\n\n
========== Defuse Phase_6 ==========

(gdb) disassemble phase_6
Dump of assembler code for function phase_6:
0x08048d98 <phase_6+0>: push %ebp
0x08048da4 <phase_6+12>: movl $0x804b26c,-0x34(%ebp)
…
0x08048db2 <phase_6+26>: push %edx
0x08048db3 <phase_6+27>: call 0x8048fd8 <read_six_numbers>
0x08048db8 <phase_6+32>: xor %edi,%edi
0x08048dba <phase_6+34>: add $0x10,%esp
0x08048dbd <phase_6+37>: lea 0x0(%esi),%esi
0x08048dc0 <phase_6+40>: lea -0x18(%ebp),%eax
0x08048dc3 <phase_6+43>: mov (%eax,%edi,4),%eax
0x08048dc6 <phase_6+46>: dec %eax
0x08048dc7 <phase_6+47>: cmp $0x5,%eax
0x08048dca <phase_6+50>: jbe 0x8048dd1 <phase_6+57>
0x08048dcc <phase_6+52>: call 0x80494fc <explode_bomb>
0x08048dd1 <phase_6+57>: lea 0x1(%edi),%ebx
0x08048dd4 <phase_6+60>: cmp $0x5,%ebx
0x08048dd7 <phase_6+63>: jg 0x8048dfc <phase_6+100>
0x08048dd9 <phase_6+65>: lea 0x0(,%edi,4),%eax
0x08048de0 <phase_6+72>: mov %eax,-0x38(%ebp)
0x08048de3 <phase_6+75>: lea -0x18(%ebp),%esi
0x08048de6 <phase_6+78>: mov -0x38(%ebp),%edx
0x08048de9 <phase_6+81>: mov (%edx,%esi,1),%eax
0x08048dec <phase_6+84>: cmp (%esi,%ebx,4),%eax
0x08048def <phase_6+87>: jne 0x8048df6 <phase_6+94>
0x08048df1 <phase_6+89>: call 0x80494fc <explode_bomb>
0x08048df6 <phase_6+94>: inc %ebx
0x08048df7 <phase_6+95>: cmp $0x5,%ebx
0x08048dfa <phase_6+98>: jle 0x8048de6 <phase_6+78>
0x08048dfc <phase_6+100>: inc %edi
0x08048dfd <phase_6+101>: cmp $0x5,%edi
0x08048e00 <phase_6+104>: jle 0x8048dc0 <phase_6+40>
0x08048e02 <phase_6+106>: xor %edi,%edi
0x08048e0d <phase_6+117>: lea 0x0(%esi),%esi

链表头指针为 0x804b26c,可以根据该指针遍历所有节点。

输入 6 个数字

保证 6 个数不同

```
For(I= 0; I <= 5; i++){
    If(a[i] >6)
        explode_bomb();
    For(j=i+1; j<=5; j++){
        If(a[i] == a[j])
            explode_bomb();
    }
```

```
0x08048e10 <phase_6+120>:mov    -0x34(%ebp),%esi
0x08048e13 <phase_6+123>:mov    $0x1,%ebx
0x08048e18 <phase_6+128>:lea    0x0(,%edi,4),%eax
0x08048e1f <phase_6+135>: mov    %eax,%edx
0x08048e21 <phase_6+137>:cmp    (%eax,%ecx,1),%ebx
0x08048e24 <phase_6+140>:jge    0x8048e38 <phase_6+160>
0x08048e26 <phase_6+142>:mov    (%edx,%ecx,1),%eax
0x08048e29 <phase_6+145>:lea    0x0(%esi,%eiz,1),%esi
0x08048e30 <phase_6+152>:mov    0x8(%esi),%esi
0x08048e33 <phase_6+155>:inc    %ebx
0x08048e34 <phase_6+156>:cmp    %eax,%ebx
0x08048e36 <phase_6+158>:jl     0x8048e30 <phase_6+152>
0x08048e38 <phase_6+160>:mov    -0x3c(%ebp),%edx
0x08048e3b <phase_6+163>:mov    %esi,(%edx,%edi,4)
0x08048e3e <phase_6+166>:inc    %edi
0x08048e3f <phase_6+167>: cmp    $0x5,%edi
0x08048e42 <phase_6+170>:jle    0x8048e10 <phase_6+120>
0x08048e44 <phase_6+172>:mov    -0x30(%ebp),%esi
0x08048e47 <phase_6+175>:mov    %esi,-0x34(%ebp)
0x08048e4a <phase_6+178>:mov    $0x1,%edi
0x08048e4f <phase_6+183>: lea    -0x30(%ebp),%edx
0x08048e52 <phase_6+186>:mov    (%edx,%edi,4),%eax
0x08048e55 <phase_6+189>:mov    %eax,0x8(%esi)
0x08048e58 <phase_6+192>:mov    %eax,%esi
0x08048e5a <phase_6+194>:inc    %edi
0x08048e5b <phase_6+195>:cmp    $0x5,%edi
0x08048e5e <phase_6+198>:jle    0x8048e52 <phase_6+186>
0x08048e60 <phase_6+200>:movl   $0x0,0x8(%esi)
0x08048e67 <phase_6+207>:mov    -0x34(%ebp),%esi
0x08048e6a <phase_6+210>:xor    %edi,%edi
0x08048e6c <phase_6+212>:lea    0x0(%esi,%eiz,1),%esi
0x08048e70 <phase_6+216>:mov    0x8(%esi),%edx
0x08048e73 <phase_6+219>:mov    (%esi),%eax
0x08048e75 <phase_6+221>:cmp    (%edx),%eax
0x08048e77 <phase_6+223>:jge    0x8048e7e <phase_6+230>
0x08048e79 <phase_6+225>:call   0x80494fc <explode_bomb>
0x08048e7e <phase_6+230>:mov    0x8(%esi),%esi
0x08048e81 <phase_6+233>:inc    %edi
0x08048e82 <phase_6+234>:cmp    $0x4,%edi
0x08048e85 <phase_6+237>:jle    0x8048e70 <phase_6+216>
0x08048e87 <phase_6+239>:lea    -0x58(%ebp),%esp
0x08048e8a <phase_6+242>:pop    %ebx
0x08048e8b <phase_6+243>:pop    %esi
0x08048e8c <phase_6+244>:pop    %edi
```

将链表 list 根据输入的 6 个数作为索引来排序,存放到 list_array 中

```
For(i=0; i<=5; i++){
        p = list;
        for(j=1; j<a[i]; j++)
                p = p->next;
        list_array[i]=p;
}
```

根据 list_array 重新建链表

判断 list_array 中的节点是否从大是从大到小排列?

```
p = list_array;
For(i=0; i<=4; i++){
        If(*p <*p->next)
                explode_bomb();
        p = p->next;
}
```

```
0x08048e8d <phase_6+245>:mov    %ebp,%esp
0x08048e8f <phase_6+247>: pop    %ebp
0x08048e90 <phase_6+248>:ret
End of assembler dump.
(gdb) x /32x 0x804b26c
0x804b26c <node1>:    0xfd 0x00    0x00    0x00    0x01    0x00    0x00    0x00
0x804b274 <node1+8>: 0x60    0xb2    0x04    0x08    0xe9 0x03    0x00    0x00
0x804b27c <n48+4>:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x804b284 <n46>: 0x2f 0x00    0x00    0x00    0x00    0x00    0x00    0x00
(gdb) x /16x 0x804b260
0x804b260 <node2>:    0xd5    0x02    0x00    0x00    0x02    0x00    0x00    0x00
0x804b268 <node2+8>: 0x54    0xb2    0x04    0x08    0xfd 0x00    0x00    0x00
(gdb) x /16x 0x804b254
0x804b254 <node3>:    0x2d    0x01    0x00    0x00    0x03    0x00    0x00    0x00
0x804b25c <node3+8>: 0x48    0xb2    0x04    0x08    0xd5    0x02    0x00    0x00
(gdb) x /16x 0x804b248
0x804b248 <node4>:    0xe5 0x03    0x00    0x00    0x04    0x00    0x00    0x00
0x804b250 <node4+8>: 0x3c 0xb2    0x04    0x08    0x2d    0x01    0x00    0x00
(gdb) x /16x 0x804b23c
0x804b23c <node5>:    0xd4    0x00    0x00    0x00    0x05    0x00    0x00    0x00
0x804b244 <node5+8>: 0x30    0xb2    0x04    0x08    0xe5 0x03    0x00    0x00
(gdb) x /16x 0x804b230
0x804b230 <node6>:    0xb0    0x01    0x00    0x00    0x06    0x00    0x00    0x00
0x804b238 <node6+8>: 0x00    0x00    0x00    0x00    0xd4    0x00    0x00    0x00
(gdb) p node1
$1 = 253
(gdb) p node2
$2 = 725
(gdb) p node3
$3 = 301
(gdb) p node4
$4 = 997
(gdb) p node5
$5 = 212
(gdb) p node6
$6 = 432
```

原始链表信息可以从这个头指针获得，我们可以知道里面存放的是”253,725,301,997,212,432”。所以按照从大到小排列对应的输入为：”4 2 6 3 1 5”

```
(gdb) echo ========================= Defuse Secret_Phase ============================\n\n
========================= Defuse Secret_Phase ============================

(gdb) disassemble phase_defused
Dump of assembler code for function phase_defused:
0x0804952c <phase_defused+0>: push   %ebp
0x0804952d <phase_defused+1>: mov    %esp,%ebp
0x0804952f <phase_defused+3>: sub    $0x64,%esp
0x08049532 <phase_defused+6>: push   %ebx
0x08049533 <phase_defused+7>: cmpl   $0x6,0x804b480
0x0804953a <phase_defused+14>:      jne    0x804959f <phase_defused+115>
0x0804953c <phase_defused+16>:      lea    -0x50(%ebp),%ebx
0x0804953f <phase_defused+19>:push   %ebx
0x08049540 <phase_defused+20>:      lea    -0x54(%ebp),%eax
0x08049543 <phase_defused+23>:      push   %eax
0x08049544 <phase_defused+24>:      push   $0x8049d03
0x08049549 <phase_defused+29>:      push   $0x804b770
0x0804954e <phase_defused+34>:      call   0x8048860 <sscanf@plt>
0x08049553 <phase_defused+39>:      add    $0x10,%esp
0x08049556 <phase_defused+42>:      cmp    $0x2,%eax
0x08049559 <phase_defused+45>:      jne    0x8049592 <phase_defused+102>
0x0804955b <phase_defused+47>:      add    $0xfffffff8,%esp
0x0804955e <phase_defused+50>:      push   $0x8049d09
0x08049563 <phase_defused+55>:      push   %ebx
0x08049564 <phase_defused+56>:      call   0x8049030 <strings_not_equal>
0x08049569 <phase_defused+61>:      add    $0x10,%esp
0x0804956c <phase_defused+64>:      test   %eax,%eax
…
0x08049585 <phase_defused+89>:      call   0x8048810 <printf@plt>
0x0804958a <phase_defused+94>:      add    $0x20,%esp
0x0804958d <phase_defused+97>:      call   0x8048ee8 <secret_phase>
0x08049592 <phase_defused+102>:     add    $0xfffffff4,%esp
0x08049595 <phase_defused+105>:     push   $0x8049da0
0x0804959a <phase_defused+110>:     call   0x8048810 <printf@plt>
0x0804959f <phase_defused+115>:     mov    -0x68(%ebp),%ebx
0x080495a2 <phase_defused+118>:     mov    %ebp,%esp
0x080495a4 <phase_defused+120>:     pop    %ebp
0x080495a5 <phase_defused+121>:     ret
End of assembler dump.
(gdb) x /32c 0x8049d03
0x8049d03:    37 '%'    100 'd'    32 ' '37 '%'    115 's'    0 '\0'97 'a'    117 'u'
0x8049d0b:    115 's'    116 't'    105 'i'    110 'n'    112 'p'    111 'o'    119 'w'    101 'e'
0x8049d13:    114 'r'    115 's'    0 '\0'0 '\0'0 '\0'0 '\0'0 '\0'0 '\0'
0x8049d1b:    0 '\0'0 '\0'0 '\0'0 '\0'0 '\0'67 'C'    117 'u'    114 'r'
```

Secret_phase 的入口隐藏在这里。从 0x8049d03 来看要求输入“%d %s”，从下面调用 strings_not_equal 来看，输入字符串应该是“austinpowers”。

但 0x804b770 不是正常的输入变量，所有有两种办法来给它填数据:一种是利用缓冲区溢出；另一种更简单粗暴的方法就是用 gdb 直接向那块内存区域写数据。这里采用了第二种。具体参见下页

输入检查通过后调用 secret_phase()

```
(gdb) x /32c 0x804b770
0x804b770 <input_strings+240>:  0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x804b778 <input_strings+248>:  0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x804b780 <input_strings+256>:  0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x804b788 <input_strings+264>:  0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0'
(gdb) x /32c 0x8049d09
0x8049d09:    97 'a'    117 'u'    115 's'    116 't'    105 'i'    110 'n'    112 'p'    111 'o'
0x8049d11:    119 'w'    101 'e'    114 'r'    115 's'    0 '\0' 0 '\0' 0 '\0' 0 '\0'
0x8049d19:    0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 0 '\0' 67 'C'
0x8049d21:    117 'u'    114 'r'    115 's'    101 'e'    115 's'    44 ',' 32 ' ' 121 'y'
(gdb) set *(char*)0x804b770 = '2'
Cannot access memory at address 0x804b770
(gdb) r
Starting program: /memex/ybao/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!

Program received signal SIGINT, Interrupt.
0xffffe410 in __kernel_vsyscall ()
(gdb) set *(char*)0x804b770 = '2'
(gdb) set *(char*)0x804b771 = ' '
(gdb) set *(char*)0x804b772 = 'a'
(gdb) set *(char*)0x804b773 = 'u'
(gdb) set *(char*)0x804b774 = 's'
(gdb) set *(char*)0x804b775 = 't'
(gdb) set *(char*)0x804b776 = 'i'
(gdb) set *(char*)0x804b777 = 'n'
(gdb) set *(char*)0x804b778 = 'p'
(gdb) set *(char*)0x804b779 = 'o'
(gdb) set *(char*)0x804b77a = 'w'
(gdb) set *(char*)0x804b77b = 'e'
(gdb) set *(char*)0x804b77c = 'r'
(gdb) set *(char*)0x804b77d = 's'
(gdb) disassemble secret_phase
Dump of assembler code for function secret_phase:
0x08048ee8 <secret_phase+0>:    push   %ebp
…
0x08048eef <secret_phase+7>:    call   0x80491fc <read_line>
0x08048ef4 <secret_phase+12>:   push   $0x0
0x08048ef6 <secret_phase+14>:   push   $0xa
0x08048ef8 <secret_phase+16>:   push   $0x0
0x08048efa <secret_phase+18>:   push   %eax
0x08048efb <secret_phase+19>:   call   0x80487f0 <__strtol_internal@plt>
0x08048f00 <secret_phase+24>:   add    $0x10,%esp
```

进入 secret_phase 后有要求输入一个数

```
0x08048f03 <secret_phase+27>:   mov      %eax,%ebx
0x08048f05 <secret_phase+29>:   lea      -0x1(%ebx),%eax
0x08048f08 <secret_phase+32>:   cmp      $0x3e8,%eax
0x08048f0d <secret_phase+37>:   jbe      0x8048f14 <secret_phase+44>
0x08048f0f <secret_phase+39>:   call     0x80494fc <explode_bomb>
0x08048f14 <secret_phase+44>:   add      $0xffffff8,%esp
0x08048f17 <secret_phase+47>:   push     %ebx
0x08048f18 <secret_phase+48>:   push     $0x804b320
0x08048f1d <secret_phase+53>:   call     0x8048e94 <fun7>
0x08048f22 <secret_phase+58>:   add      $0x10,%esp
0x08048f25 <secret_phase+61>:   cmp      $0x7,%eax
0x08048f28 <secret_phase+64>:   je       0x8048f2f <secret_phase+71>
0x08048f2a <secret_phase+66>:   call     0x80494fc <explode_bomb>
0x08048f2f <secret_phase+71>:   add      $0xffffff4,%esp
0x08048f32 <secret_phase+74>:   push     $0x8049820
0x08048f37 <secret_phase+79>:   call     0x8048810 <printf@plt>
0x08048f3c <secret_phase+84>:   call     0x804952c <phase_defused>
0x08048f41 <secret_phase+89>:   mov      -0x18(%ebp),%ebx
0x08048f44 <secret_phase+92>:   mov      %ebp,%esp
0x08048f46 <secret_phase+94>:   pop      %ebp
0x08048f47 <secret_phase+95>:   re
End of assembler dump.
(gdb) x /32x 0x804b320
0x804b320 <n1>:   0x24      0x00      0x00      0x00      0x14      0xb3      0x04      0x08
0x804b328 <n1+8>:      0x08      0xb3      0x04      0x08      0x00      0x00      0x00      0x00
0x804b330:    0x00      0x00      0x00      0x00      0x00      0x00      0x00      0x00
0x804b338:    0x00      0x00      0x00      0x00      0x00      0x00      0x00      0x00
(gdb) x /32x 0x804b314
0x804b314 <n21>: 0x08      0x00      0x00      0x00      0xe40xb2      0x04      0x08
0x804b31c <n21+8>:     0xfc 0xb2      0x04      0x08      0x24      0x00      0x00      0x00
0x804b324 <n1+4>:      0x14      0xb3      0x04      0x08      0x08      0xb3      0x04      0x08
0x804b32c:    0x00      0x00      0x00      0x00      0x00      0x00      0x00      0x00
(gdb) x /16x 0x804b308
0x804b308 <n22>: 0x32      0x00      0x00      0x00      0xf0 0xb2      0x04      0x08
0x804b310 <n22+8>:     0xd8      0xb2      0x04      0x08      0x08      0x00      0x00      0x00
(gdb) x /16x 0x804b2f0
0x804b2f0 <n33>:  0x2d      0x00      0x00      0x00      0xcc 0xb2      0x04      0x08
0x804b2f8 <n33+8>:     0x84      0xb2      0x04      0x08      0x16      0x00      0x00      0x00
(gdb) x /16x 0x804b2d8
0x804b2d8 <n34>: 0x6b      0x00      0x00      0x00      0xb4      0xb2      0x04      0x08
0x804b2e0 <n34+8>:     0x78      0xb2      0x04      0x08      0x06      0x00      0x00      0x00
(gdb) x /16x n31
0x6: Cannot access memory at address 0x6
(gdb) x /16x &n31
```

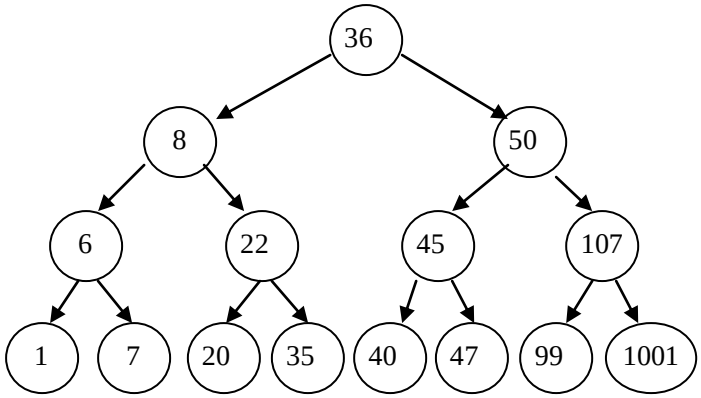
很关键，压入两个参数，然后调用 fun7(0x804b320, %ebx)

根据 func7 的操作，可以知道第一个参数是一个二叉排序树的 root，而%ebx 就是输入的数。

Func7 返回值应该是 7

有了 root，我们可以把二叉排序树恢复出来。

0x804b2e4 <n31>: 0x06 0x00 0x00 0x00 0xc00xb2 0x04 0x08
0x804b2ec <n31+8>: 0x9c0xb2 0x04 0x08 0x2d 0x00 0x00 0x00
(gdb) x /16x &n32
0x804b2fc <n32>: 0x16 0x00 0x00 0x00 0x90 0xb2 0x04 0x08
0x804b304 <n32+8>: 0xa80xb2 0x04 0x08 0x32 0x00 0x00 0x00
(gdb) info symbol
Argument required (address).
(gdb) p n1
$7 = 36
(gdb) p n21
$8 = 8
(gdb) p n22
$9 = 50
(gdb) p n31
$10 = 6
(gdb) p n32
$11 = 22
(gdb) p n33
$12 = 45
(gdb) p n34
$13 = 107
(gdb) p n41
$14 = 1
(gdb) p n42
$15 = 7
(gdb) p n43
$16 = 20
(gdb) p n44
$17 = 35
(gdb) p n45
$18 = 40
(gdb) p n46
$19 = 47
(gdb) p n47
$20 = 99
(gdb) p n48
$21 = 1001



```
(gdb) disassemble fun7
Dump of assembler code for function fun7:
0x08048e94 <fun7+0>:  push   %ebp
0x08048e95 <fun7+1>:  mov    %esp,%ebp
0x08048e97 <fun7+3>:  sub    $0x8,%esp
0x08048e9a <fun7+6>:  mov    0x8(%ebp),%edx
0x08048e9d <fun7+9>:  mov    0xc(%ebp),%eax
0x08048ea0 <fun7+12>:test    %edx,%edx
0x08048ea2 <fun7+14>:jne     0x8048eb0 <fun7+28>
0x08048ea4 <fun7+16>:mov     $0xffffffff,%eax
0x08048ea9 <fun7+21>:jmp     0x8048ee2 <fun7+78>
0x08048eab <fun7+23>:nop
0x08048eac <fun7+24>:lea     0x0(%esi,%eiz,1),%esi
0x08048eb0 <fun7+28>:cmp     (%edx),%eax
0x08048eb2 <fun7+30>:jge     0x8048ec5 <fun7+49>
0x08048eb4 <fun7+32>:add     $0xfffffff8,%esp
0x08048eb7 <fun7+35>:push    %eax
0x08048eb8 <fun7+36>:mov     0x4(%edx),%eax
0x08048ebb <fun7+39>:push    %eax
0x08048ebc <fun7+40>:call    0x8048e94 <fun7>
0x08048ec1 <fun7+45>:add     %eax,%eax
0x08048ec3 <fun7+47>:jmp     0x8048ee2 <fun7+78>
0x08048ec5 <fun7+49>:cmp     (%edx),%eax
0x08048ec7 <fun7+51>:je      0x8048ee0 <fun7+76>
0x08048ec9 <fun7+53>:add     $0xfffffff8,%esp
0x08048ecc <fun7+56>:push    %eax
0x08048ecd <fun7+57>:mov     0x8(%edx),%eax
0x08048ed0 <fun7+60>:push    %eax
0x08048ed1 <fun7+61>:call    0x8048e94 <fun7>
0x08048ed6 <fun7+66>:add     %eax,%eax
0x08048ed8 <fun7+68>:inc     %eax
0x08048ed9 <fun7+69>:jmp     0x8048ee2 <fun7+78>
0x08048edb <fun7+71>:nop
0x08048edc <fun7+72>:lea     0x0(%esi,%eiz,1),%esi
0x08048ee0 <fun7+76>:xor     %eax,%eax
0x08048ee2 <fun7+78>:mov     %ebp,%esp
0x08048ee4 <fun7+80>:pop     %ebp
0x08048ee5 <fun7+81>:ret
End of assembler dump.
```

fun7 也是一个递归函数，对应的代码是：

```
int fun7(node, value)
{
    If(node == NULL)
        return -1;

    if(value == *node)
        return 0;
    else if(value < *node)
        return 2*func7(node->left, value);
    else return 2*func7(node->right,value)+1;
}
```

因此，func7 返回值要是 7，那么可以得到 value 对应节点的值应该是 1001

所以 secret_phase 的输入时 1001