综合实验 二进制炸弹

181840326　张祎扬

一、实验目的

使用课程所学知识拆除一个“binary bombs”来增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。

二、实验内容与过程

阶段 1:字符串比较

首先观察阶段一的c代码：

```
printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
printf("which to blow yourself up. Have a nice day!\n");

/* Hmm...  Six phases must be more secure than one phase! */
input = read_line();             /* Get input                   */
phase_1(input);                  /* Run the phase               */
phase_defused();                 /* Drat!  They figured it out!
                                  * Let me know how they did it. */
printf("Phase 1 defused. How about the next one?\n");
```

发现主要由三个函数组成，即read_line(),phase_1(),和phase_defused().将可执行文件反汇编，观察这几个函数的汇编代码。

```
08048b33 <phase_1>:
 8048b33:       83 ec 14                sub    $0x14,%esp
 8048b36:       68 04 a0 04 08          push   $0x804a004
 8048b3b:       ff 74 24 1c             pushl  0x1c(%esp)
 8048b3f:       e8 aa 04 00 00          call   8048fee <strings_not_equal>
 8048b44:       83 c4 10                add    $0x10,%esp
 8048b47:       85 c0                   test   %eax,%eax
 8048b49:       74 05                   je     8048b50 <phase_1+0x1d>
 8048b4b:       e8 95 05 00 00          call   80490e5 <explode_bomb>
 8048b50:       83 c4 0c                add    $0xc,%esp
 8048b53:       c3                      ret
```

从汇编代码中可以推断，当test %eax的值为0时，将自动跳转到结束，否则将触发explode_bomb函数，炸弹爆炸。而%eax中存放的是函数strings_not_equal的返回值,所以当函数返回0，即两个字符串相等的时候炸弹破解。又根据第二行第三行，这两个地址应该存放的是输入流输入的字符串和解开炸弹的密码。据我推断，第二行的固定地址0x804a004存放的应该是解开炸弹的密码字符串。

然后我使用指令objdump -s 反汇编可执行文件，找到section .rodata,找到地址0x804a004开始存放的字符串。

```
804a000 6e652e00 466f7220 4e415341 2c207370  ne..For NASA, sp
804a010 61636520 69732073 74696c6c 20612068  ace is still a h
804a020 69676820 7072696f 72697479 2e000000  igh priority....
```

根据ascii码推断，解开阶段一的字符串是"For NASA, space is still a high priority."

运行./bomb进行尝试，发现结果正确。

```
katherine@katherine-virtual-machine:~/Desktop/181840326$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
For NASA, space is still a high priority.
Phase 1 defused. How about the next one?
```

阶段 2:循环

首先观察阶段二的c代码：

```
/* The second phase is harder.  No one will ever figure out
 * how to defuse this... */
input = read_line();
phase_2(input);
phase_defused();
printf("That's number 2.  Keep going!\n");
```

发现由read_line(),phase_2(),phase_defused()三个函数组成。下面我重点关注函数phase_2()的汇编代码。

```
08048b54 <phase_2>:
 8048b54:       56                      push   %esi
 8048b55:       53                      push   %ebx
 8048b56:       83 ec 2c                sub    $0x2c,%esp
 8048b59:       65 a1 14 00 00 00       mov    %gs:0x14,%eax
 8048b5f:       89 44 24 24             mov    %eax,0x24(%esp)
 8048b63:       31 c0                   xor    %eax,%eax
 8048b65:       8d 44 24 0c             lea    0xc(%esp),%eax
 8048b69:       50                      push   %eax
 8048b6a:       ff 74 24 3c             pushl  0x3c(%esp)
 8048b6e:       e8 97 05 00 00          call   804910a <read_six_numbers>
 8048b73:       83 c4 10                add    $0x10,%esp
 8048b76:       83 7c 24 04 00          cmpl   $0x0,0x4(%esp)
 8048b7b:       75 07                   jne    8048b84 <phase_2+0x30>
 8048b7d:       83 7c 24 08 01          cmpl   $0x1,0x8(%esp)
 8048b82:       74 05                   je     8048b89 <phase_2+0x35>
 8048b84:       e8 5c 05 00 00          call   80490e5 <explode_bomb>
 8048b89:       8d 5c 24 04             lea    0x4(%esp),%ebx
 8048b8d:       8d 74 24 14             lea    0x14(%esp),%esi
 8048b91:       8b 43 04                mov    0x4(%ebx),%eax
 8048b94:       03 03                   add    (%ebx),%eax
 8048b96:       39 43 08                cmp    %eax,0x8(%ebx)
 8048b99:       74 05                   je     8048ba0 <phase_2+0x4c>
 8048b9b:       e8 45 05 00 00          call   80490e5 <explode_bomb>
 8048ba0:       83 c3 04                add    $0x4,%ebx
 8048ba3:       39 f3                   cmp    %esi,%ebx
 8048ba5:       75 ea                   jne    8048b91 <phase_2+0x3d>
 8048ba7:       8b 44 24 1c             mov    0x1c(%esp),%eax
 8048bab:       65 33 05 14 00 00 00    xor    %gs:0x14,%eax
 8048bb2:       74 05                   je     8048bb9 <phase_2+0x65>
 8048bb4:       e8 d7 fb ff ff          call   8048790 <__stack_chk_fail@plt>
 8048bb9:       83 c4 24                add    $0x24,%esp
 8048bbc:       5b                      pop    %ebx
 8048bbd:       5e                      pop    %esi
 8048bbe:       c3                      ret
```

从call read_six_numbers可以看出，阶段二需要连续输入6个数。

从第一个跳转条件cmpl $0x0,0x4(%esp)可知，当第一个数不为0时，跳转至explode_bomb,所以第一个数是0.

从第二个跳转条件cmpl $0x1,0x8(%esp)可知，当当第二个数是1时，不触发炸弹爆炸，所以第二个数是1.

从地址[0x8048b89,0x8048b94]处指令可以看出，这几步求的是第一个数和第二个数的和，如果和与第三个数（即0x8(%ebx)）相等，则不触发炸弹，所以第三个数是第一个数和第二个数的和，即0+1=1.

add $0x4,%ebx使之指向下一个数，如果%ebx！=%esi，则继续跳转，进入循环，重复上述操作。所以可以推断这里使用循环结构求和，下一个数是前两个数的和，又因为一共输入六个数，所以根据前两个数0 1，可以推断出这里的密码序列是0 1 1 2 3 5.

运行程序，进行测试，发现结果正确。

```
katherine@katherine-virtual-machine:~/Desktop/181840326$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
For NASA, space is still a high priority.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
```

阶段 3:条件/分支

继续观察函数phase_3()的汇编代码

```
8048bf3:       83 7c 24 04 07          cmpl   $0x7,0x4(%esp)
8048bf8:       77 3c                   ja     8048c36 <phase_3+0x77>
8048bfa:       8b 44 24 04             mov    0x4(%esp),%eax
8048bfe:       ff 24 85 60 a0 04 08    jmp    *0x804a060(,%eax,4)
8048c05:       b8 44 03 00 00          mov    $0x344,%eax
8048c0a:       eb 3b                   jmp    8048c47 <phase_3+0x88>
8048c0c:       b8 7d 03 00 00          mov    $0x37d,%eax
8048c11:       eb 34                   jmp    8048c47 <phase_3+0x88>
8048c13:       b8 51 02 00 00          mov    $0x251,%eax
8048c18:       eb 2d                   jmp    8048c47 <phase_3+0x88>
8048c1a:       b8 6b 03 00 00          mov    $0x36b,%eax
8048c1f:       eb 26                   jmp    8048c47 <phase_3+0x88>
8048c21:       b8 08 02 00 00          mov    $0x208,%eax
8048c26:       eb 1f                   jmp    8048c47 <phase_3+0x88>
8048c28:       b8 e9 00 00 00          mov    $0xe9,%eax
8048c2d:       eb 18                   jmp    8048c47 <phase_3+0x88>
8048c2f:       b8 4e 01 00 00          mov    $0x14e,%eax
8048c34:       eb 11                   jmp    8048c47 <phase_3+0x88>
8048c36:       e8 aa 04 00 00          call   80490e5 <explode_bomb>
8048c3b:       b8 00 00 00 00          mov    $0x0,%eax
8048c40:       eb 05                   jmp    8048c47 <phase_3+0x88>
8048c42:       b8 74 01 00 00          mov    $0x174,%eax
8048c47:       3b 44 24 08             cmp    0x8(%esp),%eax
8048c4b:       74 05                   je     8048c52 <phase_3+0x93>
8048c4d:       e8 93 04 00 00          call   80490e5 <explode_bomb>
```

从第一行可以看出，将输入的第一个数看作无符号数，当它大于7时炸弹爆炸，所以第一个数<=7,它的范围是[0,7].

设第一个数为x，则跳转到地址0x804a060+4x.从地址[0x8048c47]处的指令cmp 0x8(%esp),%eax可以推断出，当第二个数等于寄存器%eax存放的值时，炸弹解除。中间地址[0x8048c05,0x8048c42]的指令将不同的数值存入寄存器%eax。因为x的取值范围是[0,7],所以跳转地址0x804a060+4x的取值范围为[0x804a060,0x804a07c],接下来查看这个地址区间的代码，发这一块在section .rodata区域，是跳转表（采用小端法存放），所以这一块应该是一个switch case，只要第一个数和第二个数对应符合switch case的结果应该就可以拆弹。

通过观察发现case 6的值0xe9最方便计算，是233，所以解开炸弹的输入是6 233.其他case分别为

```
case 1:836
case 2:893
case 3:593
case 4:875
case 5:520
case 6:233
case 7:334
case 0:372
```

经过验证，这些case和对应的case值都可以解开炸弹。在solution中我仍然保留的是6 和233，结果如下：

```
katherine@katherine-virtual-machine:~/Desktop/181840326$ ./bomb solution
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!
```

阶段 4:递归调用和栈

仍然观察phase_4()的汇编代码

```
8048d07:       e8 5c ff ff ff          call   8048c68 <func4>
8048d0c:       83 c4 10                add    $0x10,%esp
8048d0f:       83 f8 2b                cmp    $0x2b,%eax
8048d12:       75 07                   jne    8048d1b <phase_4+0x5a>
8048d14:       83 7c 24 08 2b          cmpl   $0x2b,0x8(%esp)
8048d19:       74 05                   je     8048d20 <phase_4+0x5f>
8048d1b:       e8 c5 03 00 00          call   80490e5 <explode_bomb>
```

首先从这段汇编代码可以看出，func4函数的返回值存放在寄存器%eax中，只有函数的返回值等于0x2b，即43时，炸弹才不会爆炸。然后比较输入的第二个数字和43，只有输入的第二个数等于43时，炸弹才不会爆炸。所以可以推断出，输入的第一个数作为函数func4的参数之一，只有当返回值为43时才可以解开炸弹，即要求使函数func4的返回值为43的参数。

```
8048cff:       6a 0e                   push   $0xe
8048d01:       6a 00                   push   $0x0
8048d03:       ff 74 24 10             pushl  0x10(%esp)
8048d07:       e8 5c ff ff ff          call   8048c68 <func4>
```

从这一段可以看出函数func4有三个参数，分别是x,0,14.下面研究函数func4的汇编代码。

```
8048c6d:    8b 54 24 10       mov    0x10(%esp),%edx     x in edx
 8048c71:   8b 74 24 14       mov    0x14(%esp),%esi     y(0) in esi
 8048c75:   8b 4c 24 18       mov    0x18(%esp),%ecx     z(14) in ecx
 8048c79:   89 c8             mov    %ecx,%eax           eax:z
 8048c7b:   29 f0             sub    %esi,%eax           eax:z-y
 8048c7d:   89 c3             mov    %eax,%ebx           ebx:z-y
 8048c7f:   c1 eb 1f          shr    $0x1f,%ebx          ebx:((z-y)>>31)
 8048c82:   01 d8             add    %ebx,%eax           eax:((z-y)>>31)+(z-y)
 8048c84:   d1 f8             sar    %eax              eax:((z-y)>>31+z-y)>>1
 8048c86:   8d 1c 30          lea    (%eax,%esi,1),%ebx ebx:int temp=((z-y)>>
                                                                  31+z-y)>>1+y
 8048c89:   39 d3             cmp    %edx,%ebx           cmp temp:x
 8048c8b:   7e 15             jle    8048ca2 <func4+0x3a>  jump:temp<=x
 8048c8d:   83 ec 04          sub    $0x4,%esp
 8048c90:   8d 43 ff          lea    -0x1(%ebx),%eax       eax:temp-1
 8048c93:   50                push   %eax
 8048c94:   56                push   %esi
 8048c95:   52                push   %edx
 8048c96:   e8 cd ff ff ff    call   8048c68 <func4>       func4(x,y,temp-1)
 8048c9b:   83 c4 10          add    $0x10,%esp
 8048c9e:   01 d8             add    %ebx,%eax        func4(x,y,temp-1)+temp
 8048ca0:   eb 19             jmp    8048cbb <func4+0x53>
 8048ca2:   89 d8             mov    %ebx,%eax             eax:temp
 8048ca4:   39 d3             cmp    %edx,%ebx            cmp temp:x
 8048ca6:   7d 13             jge    8048cbb <func4+0x53> temp>=x return temp
 8048ca8:   83 ec 04          sub    $0x4,%esp
 8048cab:   51                push   %ecx
 8048cac:   8d 43 01          lea    0x1(%ebx),%eax       eax:temp+1
 8048caf:   50                push   %eax
 8048cb0:   52                push   %edx
 8048cb1:   e8 b2 ff ff ff    call   8048c68 <func4>  return func(x,temp+1,z)
                                                              +temp
 8048cb6:   83 c4 10          add    $0x10,%esp
 8048cb9:   01 d8             add    %ebx,%eax
 8048cbb:   83 c4 04          add    $0x4,%esp
 8048cbe:   5b                pop    %ebx
 8048cbf:   5e                pop    %esi
 8048cc0:   c3                ret
```

由此可以推断出函数func4的c语言代码：

```
int func4(int x,int y,int z){
  int temp=((z-y)>>31+z-y)>>1+y;
  if(x==temp)
    return temp;
  else if(x>temp)
    return func4(x,temp+1,z)+temp;
  else
    return func4(x,y,temp-1)+temp;
}
```

```
cmpl   $0xe,0x4(%esp)
jbe    8048cfc <phase_4+0x3b>
call   80490e5 <explode_bomb>
```

以上是函数phase_4的部分汇编代码，可以看出输入的第一个整数不大于14，因此x的范围是0-14，在此范围内尝试带入不同的x值，使输出为43，最终求得x的值为12，所以这个阶段炸弹的输入是12 43。

经过验证发现是正确的。

```
katherine@katherine-virtual-machine:~/Desktop/181840326$ ./bomb solution
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.
```

阶段 5:指针

观察phase_5的汇编代码

```
8048d49:       50                      push   %eax
8048d4a:       8d 44 24 08             lea    0x8(%esp),%eax
8048d4e:       50                      push   %eax
8048d4f:       68 cf a1 04 08          push   $0x804a1cf
8048d54:       ff 74 24 2c             pushl  0x2c(%esp)
8048d58:       e8 b3 fa ff ff          call   8048810 <__isoc99_sscanf@plt>
```

查看地址0x804a1cf处的数据，可以得到这一步需读入两个整数。

```
804a1c0 702e0025 64202564 20256420 25642025  p..%d %d %d %d %
804a1d0 64202564 00457272 6f723a20 5072656d  d %d.Error: Prem
```

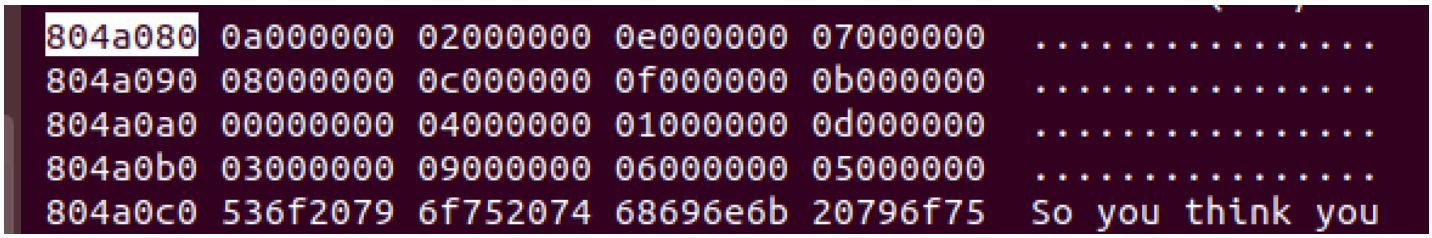
继续观察函数主体部分的汇编代码

```
8048d6a:   8b 44 24 04           mov    0x4(%esp),%eax       x->eax
8048d6e:   83 e0 0f              and    $0xf,%eax          x&1111 取x的低4位
8048d71:   89 44 24 04           mov    %eax,0x4(%esp)     x=(x&1111)
8048d75:   83 f8 0f              cmp    $0xf,%eax       cmp x:1111
8048d78:   74 2e                 je     8048da8 <phase_5+0x72> \jp x=1111
                                                                  (explode)
8048d7a:   b9 00 00 00 00        mov    $0x0,%ecx   0->ecx
8048d7f:   ba 00 00 00 00        mov    $0x0,%edx    0->edx
8048d84:   83 c2 01              add    $0x1,%edx   0+1=1->edx
8048d87:   8b 04 85 80 a0 04 08  mov    0x804a080(,%eax,4),%eax
                                                  x=*(0x804a080+4x)
8048d8e:   01 c1                 add    %eax,%ecx        x->ecx
8048d90:   83 f8 0f              cmp    $0xf,%eax      cmp x:0xf
8048d93:   75 ef                 jne    8048d84 <phase_5+0x4e>\jp x!=0xf
                                                               (循环)
8048d95:   c7 44 24 04 0f 00 00  movl   $0xf,0x4(%esp)
8048d9c:   00
8048d9d:   83 fa 0f              cmp    $0xf,%edx             edx=15
8048da0:   75 06                 jne    8048da8 <phase_5+0x72>
8048da2:   3b 4c 24 08           cmp    0x8(%esp),%ecx         ecx=y
8048da6:   74 05                 je     8048dad <phase_5+0x77>
8048da8:   e8 38 03 00 00        call   80490e5 <explode_bomb>
8048dad:   8b 44 24 0c           mov    0xc(%esp),%eax
8048db1:   65 33 05 14 00 00 00  xor    %gs:0x14,%eax
8048db8:   74 05                 je     8048dbf <phase_5+0x89>
8048dba:   e8 d1 f9 ff ff        call   8048790 <__stack_chk_fail@plt>
8048dbf:   83 c4 1c              add    $0x1c,%esp
8048dc2:   c3                    ret
```

将汇编代码写成c语言大致如下：

```c
int phase_5(int x,int y){
  temp=(x&0xf);
  if(temp==0xf)boom;
  int sum=0,i=0;
  do{
    i++;
    temp=*(0x804a080+4*temp);
    sum+=temp;
  }while(temp!=0xf);
  if(i!=15)boom;
  if(sum!=y)boom;
}
```

所以可以推断出一共进行了15次循环，而y的值就是最终的sum值。然后前往section .rodata 观察



已知一共进行了15次循环，循环结束后temp的值是15，观察此区域每4个字节出现一个数字，其他字节都是0，并且所有数字均不重复，从0-f，已知最后一次是f，所以可以倒退回第一次的值，下面以列表形式进行倒推：

数字	位次	次数
f	6	15
6	14	14
e	2	13
2	1	12
1	10	11
a	0	10
0	8	9
8	4	8
4	9	7
9	13	6
d	11	5
b	7	4
7	3	3
3	12	2
c	5	1

从这张表格所列的倒推结果可以看出，第一次循环开始时temp的值是5，所以x的值可以是5，因为y=sum，所以y的值是所有数字的和，15+6+14+2+1+10+0+8+4+9+13+11+7+3+12=115,所以求得的输入是5 115.

经过验证后发现是正确的。

```
katherine@katherine-virtual-machine:~/Desktop/181840326$ ./bomb solution
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...
```

阶段 6:链表/指针/结构

首先观察phase_6()的汇编代码，发现它同样调用了函数read_six_numbers(),所以我们知道这一步也要输入六个数字，并将它存放在一个int[6]数组内。

```
 8048de5:       be 00 00 00 00          mov    $0x0,%esi
 8048dea:       8b 44 b4 0c             mov    0xc(%esp,%esi,4),%eax
 8048dee:       83 e8 01                sub    $0x1,%eax
 8048df1:       83 f8 05                cmp    $0x5,%eax
 8048df4:       76 05                   jbe    8048dfb <phase_6+0x38>
 8048df6:       e8 ea 02 00 00          call   80490e5 <explode_bomb>
```

从这里可以看出，%esi大概率是一个计数器，它将数组中的元素放入寄存器eax中，并将这个数-1与5比较，即将这个数与6比较，如果>6,则炸弹爆炸，所以可以推断出数组中的数都不大于6.

由于这个函数的汇编代码太长，而且跳转较多，我就不在这里附上代码注释了，我会在压缩包中放一个pdf文件，里面是手写批注过的汇编代码。在这里我仅作分析。

首先代码的前半部分可以看到几个不同的循环，有大循环和小循环。计数器ebx和esi分别在两个循环中增加，每次对数组中的两个数进行比较，如果相等则爆炸。因为这两个计数器的存在，所以可以做到遍历数组中的任意两个数，所以可以得出结论，即数组中的任意两个数都不相等。

综上到目前为止的分析可以得出，数组中所有的数都不大于6，且任意两数均不相等。

继续往下分析，发现下面有循环结构。基地址主要有两个，一个是0x24(%esp),另一个是被移入寄存器edx的0x804c13c,查看这个地址的信息，发现它存储在数据段，用gdb观察如下

```
(gdb) x/20xw 0x804c13c
0x804c13c <node1>:      0x000001c6      0x00000001      0x0804c16c      0x00000355
0x804c14c <node2+4>:    0x00000002      0x0804c154      0x00000308      0x00000003
0x804c15c <node3+8>:    0x0804c160      0x00000250      0x00000004      0x0804c13c
0x804c16c <node5>:      0x000000c3      0x00000005      0x0804c178      0x00000050
0x804c17c <node6+4>:    0x00000006      0x00000000      0x0ad6a9c6      0x00000000
(gdb) shell readelf --syms bomb| grep node
    92: 0804c154    12 OBJECT  GLOBAL DEFAULT   25 node3
    94: 0804c13c    12 OBJECT  GLOBAL DEFAULT   25 node1
    98: 0804c16c    12 OBJECT  GLOBAL DEFAULT   25 node5
   147: 0804c148    12 OBJECT  GLOBAL DEFAULT   25 node2
   148: 0804c160    12 OBJECT  GLOBAL DEFAULT   25 node4
   151: 0804c178    12 OBJECT  GLOBAL DEFAULT   25 node6
(gdb)
```

可以观察到一共定义了6个node，应该是一个链表结构，每个结构的大小为12个字节。并且结构的最后一个元素是一个指针，里面存放下一个结构的地址。最后一个结构的最后一个元素是空指针。可以根据这个条件往前倒推，推出排列顺序。

地址	名称	int val	int num	Next*
0804c148	Node2	0x355	0x2	0x0804c154
0804c154	Node3	0x308	0x3	0x804c160
0x804c160	Node4	0x250	0x4	0x804c13c
0x804c13c	Node1	0x1c6	0x1	0x804c16c
0x804c16c	Node5	0xc3	0x5	0x804c178
0x804c178	Node6	0x50	0x6	0x00000000

接下来的代码定义了一个指针数组，如果a[i]<=1,那么指针数组[i]指向node1，否则指针数组[i]指向对应的node（node的编号为a[i]的值）

下一段代码的功能是按照指针数组所指结构的顺序将这些结构以链表的形式串联。最后一段代码是比较前一个结构的val值和后一个结构的val值，如果是小于，则炸弹爆炸，所以可以推断出这些结构的val值是顺序递减的，可以看出这和上述表格的顺序相符，所以最终的链表顺序如上。又因为之前分析建立的指针数组所指向的结构和一开始输入的数组是一一对应的，所以我们输入的数字是2 3 4 1 5 6.

经过验证是正确的。**还有其他一些对汇编代码的批注在压缩包的phase_6.pdf中。**

```
katherine@katherine-virtual-machine:~/Desktop/181840326$ ./bomb solution
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!
```

secret stage

首先要找出触发secret stage的输入，观察到每一个阶段完成后都有一个phase_defused()函数，用来检测是否触发了隐藏条件。

```
8049268:       68 29 a2 04 08          push   $0x804a229
804926d:       68 d0 c4 04 08          push   $0x804c4d0
8049272:       e8 99 f5 ff ff          call   8048810 <__isoc99_sscanf@plt>
8049277:       83 c4 20                add    $0x20,%esp
804927a:       83 f8 03                cmp    $0x3,%eax
804927d:       75 3a                   jne    80492b9 <phase_defused+0x7b>
804927f:       83 ec 08                sub    $0x8,%esp
8049282:       68 32 a2 04 08          push   $0x804a232
8049287:       8d 44 24 18             lea    0x18(%esp),%eax
804928b:       50                      push   %eax
804928c:       e8 5d fd ff ff          call   8048fee <strings_not_equal>
```

对应地址的数据如下：

```
804a220 63617465 642a2a2a 00256420 25642025  cated***.%d %d %
804a230 73004472 4576696c 00677265 61747768  s.DrEvil.greatwh
```

可以看出第一行push指令说明了要求输入的是两个整数和一个字符串，所以触发秘密任务的是一个字符串。根据最后一行，在调用stringsnotequal函数之前push了地址0x804a232,这个固定地址放的就是需要的字符串，再根据下面这张图发现字符串是 DrEvil 在第四个任务输入后加上DrEvil,可以看到触发了秘密任务。

```
Curses, you've found the secret phase!
But finding it and solving it are quite different...
```

然后研究函数secret_phase()的汇编代码。

```
8048f2d:       e8 77 ff ff ff          call   8048ea9 <fun7>
8048f32:       83 c4 10                add    $0x10,%esp
8048f35:       83 f8 01                cmp    $0x1,%eax
8048f38:       74 05                   je     8048f3f <secret_phase+0x45>
8048f3a:       e8 a6 01 00 00          call   80490e5 <explode_bomb>
8048f3f:       83 ec 0c                sub    $0xc,%esp
8048f42:       68 30 a0 04 08          push   $0x804a030
8048f47:       e8 74 f8 ff ff          call   80487c0 <puts@plt>
8048f4c:       e8 ed 02 00 00          call   804923e <phase_defused>
```

从这段代码可以看出，寄存器%eax中存放的是函数fun7的返回值，如果返回值不是1，炸弹将爆炸，所以需要让fun7的返回值是1.从下面的代码也可以看出fun7有两个参数，一个是我们输入的数字，另一个将该地址的数字打印出来发现是36.

```
8048f27:       53                      push   %ebx
8048f28:       68 88 c0 04 08          push   $0x804c088
8048f2d:       e8 77 ff ff ff          call   8048ea9 <fun7>
```

从汇编代码中还可以知道我们要输入一个十进制数，并且这个数字要小于等于0x3e9，即1001。继续观察fun7的汇编代码，发现它是一个递归函数，比较我们输入的值和固定地址处的值。如果输入的数等于该地址

处的数，则返回0，如果输入的数等于0，则返回-1，如果输入的数小于地址处的数，则将地址+4，返回2倍，如果输入的数大于地址处的数，则将地址+8，返回函数返回值的2倍+1.翻译成代码大致如下：

```
int fun7(addr,x){
  if(x==0)
    return -1;
  if(x==*(addr))
    return 0;
  else if(x>*(addr))
    return fun7((addr+8),x)*2+1;
  else
    return fun7((addr+4),x)*2;
}
```

已知起始地址是0x804c088,最后的返回值是1.使用gdb调试并打印不同地址处的值，可以推断出x>*(addr)情况只出现了一次，反推递归函数。因为按照正常输入，函数只可能返回0或者正数，而要求最后的返回值是1，所以只出现了一次x>*(addr)的情况，并且只可能是第一次，以后的情况只可能是x<*(addr)或者x=*(addr)，而最后一次只可能是x=*(addr)。因为已知开始的情况，所以不断尝试得到x的值为50.

验证结果是正确的。

```
katherine@katherine-virtual-machine:~/Desktop/181840326$ ./bomb solution
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...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```