

二进制炸弹实验报告

实验目的

根据二进制炸弹程序的反汇编代码，解析得到正确的输入，解开所有的炸弹。

第一阶段

第一阶段反汇编代码如下：

```
000000000001254 <phase_1>:
1254:  48 83 ec 08      sub    $0x8,%rsp
1258:  48 8d 35 ed 17 00 00  lea    0x17ed(%rip),%rsi      # 2a4c <_IO_stdin_used+0x14c>
125f:  e8 be 04 00 00      callq 1722 <strings_not_equal>
1264:  85 c0            test   %eax,%eax
1266:  75 05            jne    126d <phase_1+0x19>
1268:  48 83 c4 08      add    $0x8,%rsp
126c:  c3              retq
126d:  e8 b4 07 00 00      callq 1a26 <explode_bomb>
1272:  eb f4            jmp    1268 <phase_1+0x14>
```

第一阶段的反汇编代码中，调用了strings_not_equal函数来比较两个字符串是否相同。若相同，则通过，若不相同，则引爆炸弹。第一个字符串是用户输入的字符串，第二个字符串的地址是0x17ed(%rip)，通过在进入string_not_equal处设置断点，使用命令info registers \$rsi查看到rsi中的第二个字符串地址，使用命令x/20cb查看得到结果为“Wow! Brazil is big.”。

```
(gdb) x/20cb 0x558c089cba4c
0x558c089cba4c: 87 'W' 111 'o' 119 'w' 33 '!' 32 ' ' 66 'B' 114 'r' 97 'a'
0x558c089cba54: 122 'z' 105 'i' 108 'l' 32 ' ' 105 'i' 115 's' 32 ' ' 98 'b'
0x558c089cba5c: 105 'i' 103 'g' 46 '.' 0 '\000'
```

因此需要输入的字符串为“Wow! Brazil is big.”。

第二阶段

第二阶段主要反汇编代码如下：

```
1292:  83 3c 24 00      cmpl   $0x0,(%rsp)
1296:  75 07            jne    129f <phase_2+0x2b>
1298:  83 7c 24 04 01    cmpl   $0x1,0x4(%rsp)
129d:  74 05            je     12a4 <phase_2+0x30>
129f:  e8 82 07 00 00    callq 1a26 <explode_bomb>
12a4:  48 89 e3          mov    %rsp,%rbx
12a7:  48 8d 6b 10      lea    0x10(%rbx),%rbp
12ab:  eb 09            jmp    12b6 <phase_2+0x42>
12ad:  48 83 c3 04      add    $0x4,%rbx
12b1:  48 39 eb          cmp    %rbp,%rbx
12b4:  74 11            je     12c7 <phase_2+0x53>
12b6:  8b 43 04          mov    0x4(%rbx),%eax
12b9:  03 03            add    (%rbx),%eax
12bb:  39 43 08          cmp    %eax,0x8(%rbx)
12be:  74 ed            je     12ad <phase_2+0x39>
12c0:  e8 61 07 00 00    callq 1a26 <explode_bomb>
12c5:  eb e6            jmp    12ad <phase_2+0x39>
12c7:  48 8b 44 24 18    mov    0x18(%rsp),%rax
12cc:  64 48 33 04 25 28 00 xor    %fs:0x28,%rax
```

第二阶段的反汇编代码中，调用了read_six_numbers函数来读入6个以空格分割的整数。通过查看内存可以知道这6个整数以数组的形式储存在rsp指向的内存区域中，图中我输入了1、2、3、4、5、6，并且查看rsp指向的内存区域。

```
rsp          0x7ffc29305160    0x7ffc29305160
(gdb) x/6dw 0x7ffc29305160
0x7ffc29305160: 1      2      3      4
0x7ffc29305170: 5      6
```

接下来以arr为数组名称，同时以寄存器名称作为数组名称来分析程序跳转的规则：

- 程序首先比较了arr[0]与0的大小，如果arr[0]不为0，则跳转至分支1，否则跳转至分支2。
- 分支1引爆炸弹。
- 分支2比较arr[1]与1的大小，如果arr[1]为1，则将rsp的值赋予rbx，将rsp+0x10的值赋予rbp后跳转至分支3，否则跳转至分支1。
- 分支3将rsp的值赋予rbx，然后将rbx[0]与rbx[1]的值相加存放在eax中，然后比较arr[2]与和的大小，如果相同，则跳转至分支4，否则跳转至分支5。
- 分支4将rbx的值加上0x4，然后比较rbx与rbp的大小，如果不等，则跳转至分支3，否则跳转至分支6。
- 分支5引爆炸弹。
- 分支6检查堆栈并退出。

经过上述分析我们可以了解到，输入的6个整数有以下要求：

- arr[0] = 0
- arr[1] = 1
- arr[0] + arr[1] = arr[2]
- arr[1] + arr[2] = arr[3]
- arr[2] + arr[3] = arr[4]
- arr[3] + arr[4] = arr[5]

因此输入的6个整数为斐波那契数列0、1、1、2、3、5。

第三阶段

第三阶段主要反汇编代码如下：

```
1316: 8b 04 24      mov    (%rsp),%eax
1319: 48 8d 15 50 17 00 00    lea    0x1750(%rip),%rdx      # 2a70 <_IO_stdin_used+0x170>
1320: 48 63 04 82    movslq (%rdx,%rax,4),%rax
1324: 48 01 d0      add    %rdx,%rax
1327: ff e0        jmpq   *%rax
1329: e8 f8 06 00 00    callq 1a26 <explode_bomb>
132e: eb e0        jmp    1310 <phase_3+0x2d>
1330: b8 c9 02 00 00    mov    $0x2c9,%eax
1335: eb 3b        jmp    1372 <phase_3+0x8f>
1337: b8 cc 03 00 00    mov    $0x3cc,%eax
133c: eb 34        jmp    1372 <phase_3+0x8f>
133e: b8 45 03 00 00    mov    $0x345,%eax
1343: eb 2d        jmp    1372 <phase_3+0x8f>
1345: b8 bf 01 00 00    mov    $0x1bf,%eax
134a: eb 26        jmp    1372 <phase_3+0x8f>
134c: b8 5c 00 00 00    mov    $0x5c,%eax
1351: eb 1f        jmp    1372 <phase_3+0x8f>
1353: b8 7a 03 00 00    mov    $0x37a,%eax
1358: eb 18        jmp    1372 <phase_3+0x8f>
```

```
135a:  b8 b6 03 00 00      mov     $0x3b6,%eax
135f:  eb 11                jmp     1372 <phase_3+0x8f>
1361:  e8 c0 06 00 00      callq  1a26 <explode_bomb>
1366:  b8 00 00 00 00      mov     $0x0,%eax
136b:  eb 05                jmp     1372 <phase_3+0x8f>
136d:  b8 5a 01 00 00      mov     $0x15a,%eax
1372:  39 44 24 04          cmp     %eax,0x4(%rsp)
1376:  74 05                je      137d <phase_3+0x9a>
1378:  e8 a9 06 00 00      callq  1a26 <explode_bomb>
```

第三阶段的反汇编代码中，调用了sscanf来读入一个格式化字符串。格式化字符串的地址为0x1a0f(%rip)，通过在调用sscanf处设置断点，查看rsi中格式化字符串的地址，使用命令x/s查看到格式化字符串为“%d %d”。因此第三阶段需要读取的是两个整数，并且同样存放在以rsp为首地址的线性数组中。

```
(gdb) info r $rsi
rsi      0x56468c394d15   94861000264981
(gdb) x/s 0x56468c394d15
0x56468c394d15: "%d %d"
```

接下来程序首先检查了sscanf的返回值，如果没有读取到2个整数，则引爆炸弹。如果成功读取到两个整数，则进入switch语句。switch语句中首先检查第一个数是否大于7，否则引爆炸弹，然后根据0~7一共8个分支检查第二个数的大小。如果第二个数大小等于这个分支的要求，则通过测试，否则引爆炸弹。8个分支要求的第二个数的值如下表。

分支	第二个数的值（十六进制）
0	15A
1	2C9
2	3CC
3	345
4	1BF
5	5C
6	37A
7	3B6

只需要根据任意一个分支输入正确的数即可。我选择的是0、346。

第四阶段

第四阶段关键反汇编代码如下：

```
1403:  ba 0e 00 00 00      mov     $0xe,%edx
1408:  be 00 00 00 00      mov     $0x0,%esi
140d:  8b 3c 24              mov     (%rsp),%edi
1410:  e8 82 ff ff ff      callq  1397 <func4>
1415:  83 f8 12             cmp     $0x12,%eax
1418:  75 07                jne     1421 <phase_4+0x56>
141a:  83 7c 24 04 12      cmpl    $0x12,0x4(%rsp)
141f:  74 05                je      1426 <phase_4+0x5b>
1421:  e8 00 06 00 00      callq  1a26 <explode_bomb>
```

第四阶段反汇编代码中，同样调用了sscanf来读入一个格式化字符串。格式化字符串的地址为0x1927(%rip)，通过在调用sscanf处设置断点，查看rsi中格式化字符串的地址，使用命令x/s查看

到格式化字符串为“%d %d”。因此第三阶段需要读取的是两个整数，并且同样存放在以rsp为首地址的线性数组中。

```
(gdb) info r $rsi
rsi      0x5641715e1d15    94839074856213
(gdb) x/s 0x5641715e1d15
0x5641715e1d15: "%d %d"
```

接下来程序检查scanf的返回值，如果没有读取到2个整数，则引爆炸弹。如果成功读取到两个整数，则判断第一个整数的值是否小于等于0x0e，若不是则引爆炸弹，否则初始化参数并调用func4函数。从phase_4调用func4的第一个参数为edi，值为rsp，第二个参数为esi，值为0，第三个参数为edx，值为0x0e。func4函数返回后，程序检查返回值是否是0x12，如果不是，则引爆炸弹，否则检查4(%rsp)的值。如果4(%rsp)为0x12，则程序返回，否则引爆炸弹。

func4函数先将rbx入栈，将返回值eax设置为第三个参数edx，然后使用edx减去esi，存储在eax和ebx中。接下来将ebx中的值逻辑右移31位的结果，即ebx的符号位，加到ebx上。然后将ebx中的值算数右移1位，加上esi，并与第一个参数edi相比。如果ebx大于edi，则跳转至分支1；如果ebx小于edi，则跳转至分支2；如果ebx等于edi，则将ebx中的数送入eax并返回。分支1将-1(%rbx)送入edx，再次调用func4，将返回值与ebx相加后返回。分支2将1(%rbx)送入edx，再次调用func4，将返回值与ebx相加后返回。

具体到phase_4对func4的第一次调用中，与输入的第一个数edi比较的是0x7，此时rbx也为0x7，如果edx大于0x7，则进入第二次调用func4。第二次调用func4时，如果(edx-1)>>1等于6，则函数返回edx-0x1+0x6。因此要使函数的返回值是0x12，则输入的第一个数应该是11，输入的第二个数是18。

经过上面的分析，输入的数应该是11、18。

第五阶段

第五阶段关键反汇编代码如下：

```
1458: e8 a8 02 00 00    callq 1705 <string_length>
145d: 83 f8 06          cmp    $0x6,%eax
1460: 75 55             jne    14b7 <phase_5+0x77>
1462: b8 00 00 00 00    mov    $0x0,%eax
1467: 48 8d 0d 22 16 00 00 lea     0x1622(%rip),%rcx    # 2a90 <array.3417>
146e: 0f b6 14 03       movzbl (%rbx,%rax,1),%edx
1472: 83 e2 0f          and    $0xf,%edx
1475: 0f b6 14 11       movzbl (%rcx,%rdx,1),%edx
1479: 88 54 04 01       mov    %dl,0x1(%rsp,%rax,1)
147d: 48 83 c0 01       add    $0x1,%rax
1481: 48 83 f8 06       cmp    $0x6,%rax
1485: 75 e7             jne    146e <phase_5+0x2e>
1487: c6 44 24 07 00    movb   $0x0,0x7(%rsp)
148c: 48 8d 7c 24 01    lea     0x1(%rsp),%rdi
1491: 48 8d 35 c8 15 00 00 lea     0x15c8(%rip),%rsi    # 2a60 <_IO_stdin_used+0x160>
1498: e8 85 02 00 00    callq 1722 <strings_not_equal>
149d: 85 c0             test   %eax,%eax
149f: 75 1d             jne    14be <phase_5+0x7e>
14a1: 48 8b 44 24 08    mov     0x8(%rsp),%rax
14a6: 64 48 33 04 25 28 00 xor     %fs:0x28,%rax
14ad: 00 00
14af: 75 14             jne    14c5 <phase_5+0x85>
14b1: 48 83 c4 10       add    $0x10,%rsp
14b5: 5b               pop    %rbx
14b6: c3               retq
```

第五阶段反汇编代码中，先调用了string_length函数，并且将返回值与0x6相比较。如果返回值不是6，则引爆炸弹，否则对输入的字符串进行处理。程序首先将rcx的值赋为0x1622(%rip)，通过x/s命令查看其指向的第一个字符串的内容是“maduiersnfotvbylWow! You've defused the secret stage!”。然后程序根据输入的字符的ascii码取低四位的结果为索引，将第一个字符串中的字符组成第二个字符串，最后将第二个字符串与第三个字符串相比较。第三个字符串的内容是“flames”，索引为9、15、1、0、5、7。如果第二个字符串与第三个字符串相等，则退出，否则引爆炸弹。

对照ascii码表，可以将“9?1057”作为输入。同时我也知道了题目中有着secret stage的存在。

第六阶段

第六阶段反汇编代码中，先调用了read_six_numbers函数，将读取的数存储在以rsp为首地址的线性数组arr中，并且将r13d赋值为1，将r12、rsi、rbp都赋值为rsp。然后程序检查read_six_numbers的返回值，如果其减去1后大于5则引爆炸弹。如果正确输入6个整数，程序检查r13d的值，如果等于6则进入分支1，否则进入分支2。分支2中，对rbp指向的值即arr[0]分别与arr[1]、arr[2]、arr[3]、arr[4]、arr[5]进行比较，如果相等，则引爆炸弹，否则将ebx加上1，与5进行比较。可以推断出这里是一个二重循环，对读入的6个数进行检查，确认两两之间不相等，同时要小于等于6。因此输入的6个数为1、2、3、4、5、6的某种排列。

二重循环执行完后的程序，将ecx赋值为arr[esi]，若ecx大于1，则将0x8(%rdx)的数赋予rdx，并将eax加上1，将ecx与eax比较。重复这个过程，直到eax与ecx相等。这个过程相当于是找到输入的数中大于1的数n，并且在node1为首结点的链表上走到下一个结点，直到第n个结点。当ecx是1时，将链表首结点的值储存到以arr在第5个元素后面的区域。因此这相当于取出了链表node1[n]的地址。而实际上通过查看内存，可以得到这个链表结点指向的整数分别为337、500、829、827、613、652。

将链表结点地址储存到arr在第5个元素后面的区域后，设输入的数分别为a、b、c、d、e、f，则程序首先将node1[b]结点作为node1[a]结点的后继结点，将node1[c]作为node1[b]的后继结点，将node1[e]作为node1[c]的后继结点，将node1[f]作为node1[e]的后继结点，将node1[f]的后继结点置空。最后程序检查重新排列后的结点储存的数值是否降序排列，如果是则退出，如果不是则引爆炸弹。

因此正确的输入为1、2、5、6、4、3。

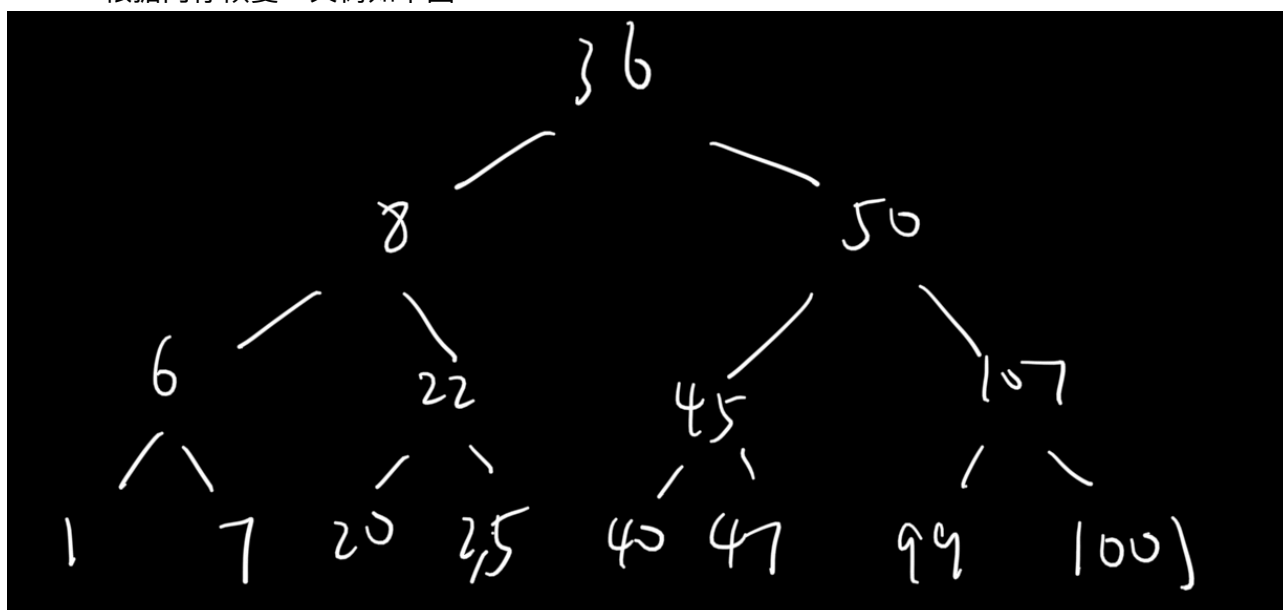
隐藏阶段

我在phase_defused中发现，要进入secret_phase，还需要输入两个整数与一个字符串，若这个字符串与“DrEvil”相等，才可以进入。通过设置断点，我发现phase_defused中读取的字符串为第四阶段输入的字符串，因此应该在第四阶段输入打开隐藏阶段的字符串，这个字符串由两个整数与一个字符串组成，前两个整数位第四阶段的答案，并且第三个字符串为“DrEvil”。

在隐藏阶段中需要输入字符串，并且调用strtol函数转换成十进制数，将结果减去1。如果减去后的结果大于1000，则引爆炸弹。如果不大于1000，调用fun7。fun7会将第二个参数esi与0x202af7(%rip)所指整数相比，通过查看内存得到这个数为36。如果esi小于rdi所指整数，则会将rdi加上8，调用fun7，将返回值乘以2返回。如果esi不等于（即大于）rdi所指整数，则会将返回值置0，调用fun7，将返回值乘以2后加上1返回。而在secret_phase中，要求fun7的返回值是6。因此在fun7被调用时，需要一次等于，两次大于，一次小于即可。查看0x202af7(%rip)所指整数，这应该是一个二叉树的结构。打印出二叉树的值如下图：

```
(gdb) x/100xw 0x55e88bb39150
0x55e88bb39150 <n1>: 0x00000024 0x00000000 0x8bb39170 0x000055e8
0x55e88bb39160 <n1+16>: 0x8bb39190 0x000055e8 0x00000000 0x00000000
0x55e88bb39170 <n21>: 0x00000008 0x00000000 0x8bb391f0 0x000055e8
0x55e88bb39180 <n21+16>: 0x8bb391b0 0x000055e8 0x00000000 0x00000000
0x55e88bb39190 <n22>: 0x00000032 0x00000000 0x8bb391d0 0x000055e8
0x55e88bb391a0 <n22+16>: 0x8bb39210 0x000055e8 0x00000000 0x00000000
0x55e88bb391b0 <n32>: 0x00000016 0x00000000 0x8bb390b0 0x000055e8
0x55e88bb391c0 <n32+16>: 0x8bb39070 0x000055e8 0x00000000 0x00000000
0x55e88bb391d0 <n33>: 0x0000002d 0x00000000 0x8bb39010 0x000055e8
0x55e88bb391e0 <n33+16>: 0x8bb390d0 0x000055e8 0x00000000 0x00000000
0x55e88bb391f0 <n31>: 0x00000006 0x00000000 0x8bb39030 0x000055e8
0x55e88bb39200 <n31+16>: 0x8bb39090 0x000055e8 0x00000000 0x00000000
---Type <return> to continue, or q <return> to quit---
0x55e88bb39210 <n34>: 0x0000006b 0x00000000 0x8bb39050 0x000055e8
0x55e88bb39220 <n34+16>: 0x8bb390f0 0x000055e8 0x00000000 0x00000000
```

根据内存恢复二叉树如下图：



由于函数需要返回6，根据二叉树按照结点大小查找的规律和上面的分析，我们的答案应该是35。

每阶段代码逻辑

第一阶段：

```
void phase_1(char* input) {
    char* answer = "Wow! Brazil is big.";

    if (strings_not_equal(input, answer)) {
        explode_bomb();
    }
}
```


第二阶段:

```
void phase_2(char* input) {
    int arr[6];
    read_six_numbers(input, arr);

    if (arr[0] != 0) {
        explode_bomb();
    }

    if (arr[1] != 1) {
        explode_bomb();
    }

    for (int i = 0; i < 4; i++) {
        if (arr[i] + arr[i+1] != arr[i+2]) {
            explode_bomb();
        }
    }
}
```

第三阶段:

```
void phase_3(char* input) {
    int a, b;

    sscanf(input, "%d %d", a, b);
    if (a > 7) {
        explode_bomb();
    }

    switch (x) {
        case 1:
            if (b != 0x2c9) {
                explode_bomb();
            }
            break;
        case 2:
            if (b != 0x3cc) {
                explode_bomb();
            }
            break;
        case 3:
            if (b != 0x345) {
                explode_bomb();
            }
            break;
        case 4:
            if (b != 0x1bf) {
                explode_bomb();
            }
            break;
        case 5:
            if (b != 0x5c) {
                explode_bomb();
            }
            break;
    }
```

```
        case 6:
            if (b != 0x37a) {
                explode_bomb();
            }
            break;
        case 7:
            if (b != 0x3b6) {
                explode_bomb();
            }
            break;
        default:
            if (b != 0x15a) {
                explode_bomb();
            }
        }
    }
}
```

第四阶段:

```
void phase_4(char* input) {
    int a, b;

    sscanf(input, "%d %d", &a, &b);

    int r = func4(a, 0, 14);

    if (r != 18) {
        explode_bomb();
    }

    if (b != 18) {
        explode_bomb();
    }
}
```

第五阶段:

```
void phase_5(char* input) {
    if (strlen(input) != 6) {
        explode_bomb();
    }

    char* table = "maduiersnfotvbyl";
    char* absver = "flames";
    char new_str[6];

    for (int i = 0; i < 6; i++) {
        new_str[i] = table[input[i] % 16];
    }

    if (!strings_not_equal(new_str, answer)) {
        explode_bomb();
    }
}
```


第六阶段:

```
struct Node {
    int num;
    Node* next;
};

void phase_6(char* input) {
    int arr[6];
    int n = read_six_numbers(input, arr);

    if (n > 6) {
        explode_bomb();
    }

    for (int i = 0; i < 6; i++) {
        if (arr[i] < 1 || arr[i] > 6) {
            explode_bomb();
        }

        for (int j = 0; j < i; j++) {
            if (arr[i] == arr[j]) {
                explode_bomb();
            }
        }
    }

    Node nodes[6];
    Node* node = nullptr;
    for (int i = 0; i < 6; i++) {
        // node1为预先定义好的链表表头
        node = node1;
        for (int j = 1; j < arr[i]; j++) {
            node = node->next;
        }
        nodes[i] = node;
    }
    for (int i = 0; i < 5; i++) {
        nodes[i+1]->next = nodes[i];
    }
    nodes[5]->next = nullptr;

    for (int i = 0; i < 5; i++) {
        if (nodes[5-i]->num > arr[4-i]) {
            explode_bomb();
        }
    }
}
```

隐藏阶段:

```
struct TreeNode {
    int num;
    TreeNode* leftChild;
    TreeNode* rightChild;
};
```

```
void secret_phase(char* input) {
    int a;

    sscanf(input, "%d", &a);

    // n1为预先定义好的二叉树根结点
    TreeNode* node = n1;

    if (fun7(a, node) != 6) {
        explode_bomb();
    }
}

int fun7(int a, TreeNode* node) {
    if (node == nullptr) {
        return 0;
    }

    if (a < node->num) {
        return 2 * fun7(a, node->leftChild);
    }
    else if (a > node->num) {
        return 2 * fun7(a, node->rightChild) +
    }
    else {
        return 0;
    }
}
```

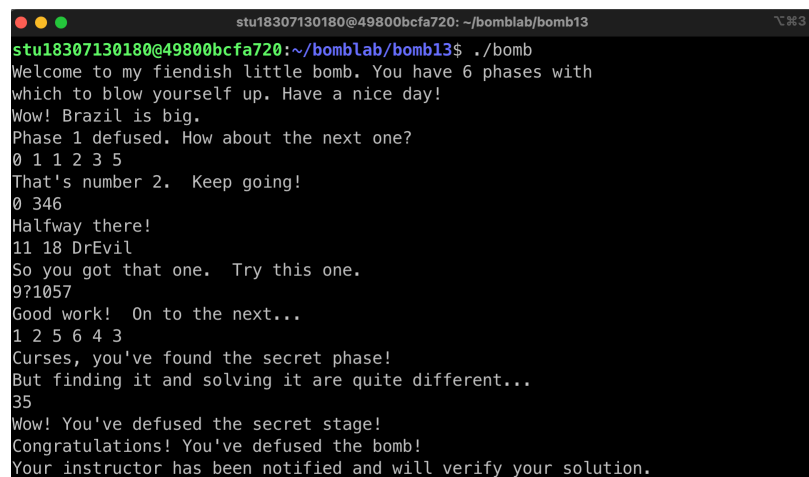
总结

最终完成的答案为：

- “Wow! Brazil is big.”
- “0 1 1 2 3 5”
- “0 346”
- “11 18 DrEvil”
- “9?1057”
- “1 2 5 6 4 3”
- “35”

在拆弹的过程中，我掌握了gdb的用法，也认识到了数组、链表、二叉树在内存中储存的方式，学习到很多东西。

最后完成的截图如下：



```
stu18307130180@49800bcfa720: ~/bomblab/bomb13
stu18307130180@49800bcfa720:~/bomblab/bomb13$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Wow! Brazil is big.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 346
Halfway there!
11 18 DrEvil
So you got that one. Try this one.
9?1057
Good work! On to the next...
1 2 5 6 4 3
Curses, you've found the secret phase!
But finding it and solving it are quite different...
35
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
```