Bomblab 实验报告

李梓童 2017202121

一、实验目标

基本目标：解开可执行文件 bomb 包括的 6 道密码。

拓展目标：解开隐藏关卡包括的密码。

二、实验方法

登录 Linux 服务器，使用 GDB 调试 bomb 可执行文件，通过分析每一段汇编代码、设置断点、运行至断点、查看寄存器和内存等方法解密。

三、实验结果

拆弹程度：解开显式关卡的 6 道密码以及隐藏关卡的密码。

密码依次为：

①I turned the moon into something I call a Death Star.

②1 2 4 8 16 32

③4 0

④108 2 DrEvil（“DrEvil”为进入隐藏关卡触发密码）

⑤pppbbc

⑥6 3 2 4 5 1

⑦（隐藏关卡）35

四、详细过程

>>第一关

```
0000000000400ef0 <phase_1>:
  400ef0:   48 83 ec 08            sub    $0x8,%rsp
# 堆栈指针前进 8 个单位
  400ef4:   be c0 24 40 00         mov    $0x4024c0,%esi
# 把某个值传入第一个参数，其值为
(gdb) x/s 0x4024c0
0x4024c0:       "I turned the moon into something I call a Death Star."
  400ef9:   e8 10 04 00 00         callq  40130e <strings_not_equal>
# 调用函数，判断字符串是否相等
  400efe:   85 c0                  test   %eax,%eax
# 判断函数返回值是否为 0
  400f00:   74 05                  je     400f07 <phase_1+0x17>
# 如果返回值为 0，则顺利退出
  400f02:   e8 6d 06 00 00         callq  401574 <explode_bomb>
# 否则炸弹爆炸
  400f07:   48 83 c4 08            add    $0x8,%rsp
# 释放临时存储空间
```

```
  400f0b:   c3                      retq
```

由调用函数<strings_not_equal>推测我们应当输入一个字符串。往上看，%esi 中是第二个参数，那么第一个参数便是我们要输入的字符串了。之后的“test %eax,%eax”是比较函数的返回值是否为 0，如果为 0 则跳转，否则炸弹爆炸。

>>第二关

```
0000000000400f0c <phase_2>:
  400f0c:   55                      push   %rbp
  400f0d:   53                      push   %rbx
  400f0e:   48 83 ec 28             sub    $0x28,%rsp
  400f12:   48 89 e6                mov    %rsp,%rsi
  400f15:   e8 90 06 00 00          callq  4015aa <read_six_numbers>
# 此处读取六个数字
  400f1a:   83 3c 24 01             cmpl   $0x1,(%rsp)
# 比较第一个数的值与 1 是否相等
  400f1e:   74 20                   je     400f40 <phase_2+0x34>
# 如果相等则跳转
  400f20:   e8 4f 06 00 00          callq  401574 <explode_bomb>
# 第一个数与 1 不相等则爆炸，此处推断第一个数为 1
  400f25:   eb 19                   jmp    400f40 <phase_2+0x34>
  400f27:   8b 43 fc                mov    -0x4(%rbx),%eax
# 把上一个数赋值给%eax
  400f2a:   01 c0                   add    %eax,%eax
# %eax 自乘 2，这是此后每次循环都进行的改变
  400f2c:   39 03                   cmp    %eax,(%rbx)
# 把%rbx 的值和%eax 比较，当%eax 里存的是第一个数时，%rbx 里存着第二个数，故推断第二个数是 1*2=2，第三个数是 2*2=4,第四个数是 4*2=8，以此类推。
  400f2e:   74 05                   je     400f35 <phase_2+0x29>
  400f30:   e8 3f 06 00 00          callq  401574 <explode_bomb>
  400f35:   48 83 c3 04             add    $0x4,%rbx
# 如果当前比较的数匹配成功，%rbx 指向下一个数
  400f39:   48 39 eb                cmp    %rbp,%rbx
  400f3c:   75 e9                   jne    400f27 <phase_2+0x1b>
# 回到循环开头
  400f3e:   eb 0c                   jmp    400f4c <phase_2+0x40>
# 如果%rbp 和%rbx 相等，则预备退出，此处是循环的结束标志
  400f40:   48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f45:   48 8d 6c 24 18          lea    0x18(%rsp),%rbp
# 初步赋值，%rbx=0x4+%rsp，%%rbp=0x18+%rsp （地址）
  400f4a:   eb db                   jmp    400f27 <phase_2+0x1b>
  400f4c:   48 83 c4 28             add    $0x28,%rsp
```

```
  400f50:   5b                      pop    %rbx
  400f51:   5d                      pop    %rbp
  400f52:   c3                      retq
```

phase_2 的代码主要包含了一个循环。输入 6 个数，其中第 1 个数是 1，然后每次执行循环的时候，都把(%rbx)里的前一个数赋值给%eax，再通过%eax 的自加，实现数字的乘 2，从而推断出数与数之间应当是 2 倍关系。最后当%rbp 和%rsp 相等时，循环结束。由初始条件可知 0x18(%rsp)-0x4(%rsp)之间有 20 个单位，恰好容纳下 5 个数，也符合题意。

>>第三关

```
0000000000400f53 <phase_3>:
  400f53:   48 83 ec 18             sub    $0x18,%rsp
  400f57:   48 8d 4c 24 08          lea    0x8(%rsp),%rcx
# 传递参数
  400f5c:   48 8d 54 24 0c          lea    0xc(%rsp),%rdx
# 传递参数
  400f61:   be ed 27 40 00          mov    $0x4027ed,%esi
# 此处 0x4027ed 的值如下：
(gdb) x/s 0x4027ed
0x4027ed:       "%d %d"
  确定了应当输入的格式
  400f66:   b8 00 00 00 00          mov    $0x0,%eax
  400f6b:   e8 c0 fc ff ff          callq  400c30 <__isoc99_sscanf@plt>
# 根据函数名中的 sscanf 和之前 0x4027ed 的值推断，这个函数将我们输入的字符串转化为两个十进制数
  400f70:   83 f8 01                cmp    $0x1,%eax
# %eax 为函数<__isoc99_sscanf@plt>的返回值，其值应当大于 1
  400f73:   7f 05                   jg     400f7a <phase_3+0x27>
  400f75:   e8 fa 05 00 00          callq  401574 <explode_bomb>
  400f7a:   83 7c 24 0c 07          cmpl   $0x7,0xc(%rsp)
# 输入的第一个参数如果比 7 大，则跳转到炸弹爆炸语句，故第一个参数不大于 7
  400f7f:   77 66                   ja     400fe7 <phase_3+0x94>
  400f81:   8b 44 24 0c             mov    0xc(%rsp),%eax
# 把第一个数的值赋给%eax
  400f85:   ff 24 c5 20 25 40 00    jmpq   *0x402520(,%rax,8)
# 关键跳转语句，内存 0x402520（可查看知道其值为 0x400f93）为起始地址，%rax 为偏移。也就是说，这里根据用户输入的参数的值来决定跳转到哪个语句，其中该参数不大于 7。假设输入的第一个参数为 4，则跳转到 0x400fb7 语句。
(gdb) x/4x 0x402520
0x402520:       0x93    0x0f    0x40    0x00
  400f8c:   b8 00 00 00 00          mov    $0x0,%eax
  400f91:   eb 05                   jmp    400f98 <phase_3+0x45>
```

```
  400f93:   b8 5c 03 00 00          mov    $0x35c,%eax
  400f98:   2d a6 03 00 00          sub    $0x3a6,%eax
  400f9d:   eb 05                   jmp    400fa4 <phase_3+0x51>
  400f9f:   b8 00 00 00 00          mov    $0x0,%eax
  400fa4:   05 03 01 00 00          add    $0x103,%eax
  400fa9:   eb 05                   jmp    400fb0 <phase_3+0x5d>
  400fab:   b8 00 00 00 00          mov    $0x0,%eax
  400fb0:   2d 69 02 00 00          sub    $0x269,%eax
  400fb5:   eb 05                   jmp    400fbc <phase_3+0x69>
  400fb7:   b8 00 00 00 00          mov    $0x0,%eax
# 跳转到该语句
  400fbc:   05 69 02 00 00          add    $0x269,%eax
  400fc1:   eb 05                   jmp    400fc8 <phase_3+0x75>
# %eax 自加 269 后跳转
  400fc3:   b8 00 00 00 00          mov    $0x0,%eax
  400fc8:   2d 69 02 00 00          sub    $0x269,%eax
  400fcd:   eb 05                   jmp    400fd4 <phase_3+0x81>
# %eax 自减 269 后跳转
  400fcf:   b8 00 00 00 00          mov    $0x0,%eax
  400fd4:   05 69 02 00 00          add    $0x269,%eax
  400fd9:   eb 05                   jmp    400fe0 <phase_3+0x8d>
# %eax 自加 269 后跳转
  400fdb:   b8 00 00 00 00          mov    $0x0,%eax
  400fe0:   2d 69 02 00 00          sub    $0x269,%eax
  400fe5:   eb 0a                   jmp    400ff1 <phase_3+0x9e>
# %eax 自减 269 后跳转
  400fe7:   e8 88 05 00 00          callq  401574 <explode_bomb>
  400fec:   b8 00 00 00 00          mov    $0x0,%eax
  400ff1:   83 7c 24 0c 05          cmpl   $0x5,0xc(%rsp)
  400ff6:   7f 06                   jg     400ffe <phase_3+0xab>
  400ff8:   3b 44 24 08             cmp    0x8(%rsp),%eax
# 最终%eax 值和第二个参数比较，如果相等，则顺利退出。根据上述逻辑，输入的第一个参数若为 4，则第二个参数为 0，这样可以得到一组答案。
  400ffc:   74 05                   je     401003 <phase_3+0xb0>
  400ffe:   e8 71 05 00 00          callq  401574 <explode_bomb>
  401003:   48 83 c4 18             add    $0x18,%rsp
  401007:   c3                      retq
```

汇编语句的功能是根据输入的第一个参数跳转到相应的赋值语句，然后从该赋值语句往下走，进行自加或自减，得到第二个参数应有的值。本题答案不唯一，除了“4 0”外，还有“1 -1292”“2 -358”“3 -617”等。

>>第四关

```
0000000000401040 <phase_4>:
  401040:   48 83 ec 18             sub    $0x18,%rsp
  401044:   48 8d 4c 24 0c          lea    0xc(%rsp),%rcx
# 输入第二个数
  401049:   48 8d 54 24 08          lea    0x8(%rsp),%rdx
# 输入的第一个数
  40104e:   be ed 27 40 00          mov    $0x4027ed,%esi
# 0x4027ed 值为“%d %d”
(gdb) x/s 0x4027ed
0x4027ed:       "%d %d"
  401053:   b8 00 00 00 00          mov    $0x0,%eax
  401058:   e8 d3 fb ff ff          callq  400c30 <__isoc99_sscanf@plt>
# 调用函数 sscanf，可推断和 phase_3 一样，要用户输入按照“%d %d”格式输入两个十进制数。
  40105d:   83 f8 02                cmp    $0x2,%eax
  401060:   75 0c                   jne    40106e <phase_4+0x2e>
  401062:   8b 44 24 0c             mov    0xc(%rsp),%eax
  401066:   83 e8 02                sub    $0x2,%eax
  401069:   83 f8 02                cmp    $0x2,%eax
  40106c:   76 05                   jbe    401073 <phase_4+0x33>
# 以上四句说明若输入的第二个数小于等于 4，则跳转到 0x401073 语句，即第二个数应当小于等于 4.
  40106e:   e8 01 05 00 00          callq  401574 <explode_bomb>
  401073:   8b 74 24 0c             mov    0xc(%rsp),%esi
# 把第二个数作为 func4 的第二个参数
  401077:   bf 08 00 00 00          mov    $0x8,%edi
# func4 第一个参数为 8
  40107c:   e8 87 ff ff ff          callq  401008 <func4>
# 调用函数 func4。根据上面 func4 代码，可知 func4 有两个参数。
  401081:   3b 44 24 08             cmp    0x8(%rsp),%eax
# 函数 func4 返回值应当和输入的第一个数相等
  401085:   74 05                   je     40108c <phase_4+0x4c>
  401087:   e8 e8 04 00 00          callq  401574 <explode_bomb>
  40108c:   48 83 c4 18             add    $0x18,%rsp
  401090:   c3                      retq
```

第四关的代码功能可以总结为：用户按照“%d %d”的格式输入数字 a b，使得 func4(8,b)=a。接下来看 func4 的代码。

```
0000000000401008 <func4>:
  401008:   41 54                   push   %r12
  40100a:   55                      push   %rbp
  40100b:   53                      push   %rbx
  40100c:   89 fb                   mov    %edi,%ebx
```

```
  40100e:   85 ff                   test   %edi,%edi
  401010:   7e 24                   jle    401036 <func4+0x2e>
# func4 的结束条件 1：第一个参数为 0，返回值为 0
  401012:   89 f5                   mov    %esi,%ebp
  401014:   89 f0                   mov    %esi,%eax
# 把第二个参数的值保存到%eax 中
  401016:   83 ff 01                cmp    $0x1,%edi
  401019:   74 20                   je     40103b <func4+0x33>
# func4 的结束条件 2：第一个参数为 1，返回值为第二个参数
  40101b:   8d 7f ff                lea    -0x1(%rdi),%edi
# 第一个参数自减 1 后作为新参数传入递归函数
  40101e:   e8 e5 ff ff ff          callq  401008 <func4>
# 递归调用函数 func4
  401023:   44 8d 24 28             lea    (%rax,%rbp,1),%r12d
# 保存第一个调用递归函数的结果
  401027:   8d 7b fe                lea    -0x2(%rbx),%edi
# 第一个参数自减 2 后作为新参数传入递归函数
  40102a:   89 ee                   mov    %ebp,%esi
  40102c:   e8 d7 ff ff ff          callq  401008 <func4>
# 第二次调用递归函数
  401031:   44 01 e0                add    %r12d,%eax
# 两次调用递归函数结果相加作为返回值
  401034:   eb 05                   jmp    40103b <func4+0x33>
  401036:   b8 00 00 00 00          mov    $0x0,%eax
  40103b:   5b                      pop    %rbx
  40103c:   5d                      pop    %rbp
  40103d:   41 5c                   pop    %r12
  40103f:   c3                      retq
```

函数 func4 可以用下列 C 语言代码表示：

```
int func4(x, y){
    if (x == 0) return 0;
    if (x == 1) return y;
    return func4(x-1,y)+func4(x-2,y)+y;
}
```

要让 y<=4，不妨取 y=2，则 func4(8,2)=108，可得一组答案 108 2（答案不唯一）.

>>第五关

```
0000000000401091 <phase_5>:
  401091:   53                      push   %rbx
  401092:   48 89 fb                mov    %rdi,%rbx
  401095:   e8 57 02 00 00          callq  4012f1 <string_length>
```

```
# 调用一个返回字符串长度的函数
  40109a:   83 f8 06                cmp    $0x6,%eax
  40109d:   74 05                   je     4010a4 <phase_5+0x13>
# 若字符串长度不为6，则爆炸。故本题答案是一个长度为6的字符串
  40109f:   e8 d0 04 00 00          callq  401574 <explode_bomb>
  4010a4:   b8 00 00 00 00          mov    $0x0,%eax
  4010a9:   ba 00 00 00 00          mov    $0x0,%edx
  4010ae:   0f b6 0c 03             movzbl (%rbx,%rax,1),%ecx
# 依次取出我们输入字符串中的字符，本质是取出了一个八位的二进制数
  4010b2:   83 e1 0f                and    $0xf,%ecx
# 保留取出字符中最后四位
  4010b5:   03 14 8d 60 25 40 00    add    0x402560(,%rcx,4),%edx
# 0x402560 内容如下：
```

```
(gdb) x/6wd 0x402560
0x402560 <array.3159>:  2       10      6       1
0x402570 <array.3159+16>:       12      16
(gdb)
```

该地址里放了一个数组，{2,10,6,1,12,16}，此处是依次取出该数组里的值累加到%edx 里。

```
  4010bc:   48 83 c0 01             add    $0x1,%rax
# %rax 自加 1，即下次取字符串里的字符时往后取一个
  4010c0:   48 83 f8 06             cmp    $0x6,%rax
  4010c4:   75 e8                   jne    4010ae <phase_5+0x1d>
# 取 6 次后结束，刚好字符串的长度为 6
  4010c6:   83 fa 13                cmp    $0x13,%edx
# 最后%edx 里的和为 13
  4010c9:   74 05                   je     4010d0 <phase_5+0x3f>
  4010cb:   e8 a4 04 00 00          callq  401574 <explode_bomb>
  4010d0:   5b                      pop    %rbx
  4010d1:   c3                      retq
```

该段汇编代码的功能：用户输入六个字符，每次取一个字符的后四位得到一个整数，该整数作为 0x402560 里存放的数组的数组下标，可以取出数组里的一个数。这样取到 6 个数，最终要使这六个数的和为 0x13=19.

令 19=2+2+2+6+6+1，相应的数组下标为 0,0,0，2,2,3，对应的二进制码有 0000,0010,0011，故可取字符 pppbbc，其中 p 的二进制码为 01000000，b 的二进制码为 01100010，c 的二进制码为 00000011。本题答案也不唯一。

>>第六关

```
00000000004010d2 <phase_6>:
  4010d2:   41 56                   push   %r14
  4010d4:   41 55                   push   %r13
  4010d6:   41 54                   push   %r12
```

```
  4010d8:   55                      push   %rbp
  4010d9:   53                      push   %rbx
  4010da:   48 83 ec 50             sub    $0x50,%rsp
# 分配局部存储空间
  4010de:   4c 8d 6c 24 30          lea    0x30(%rsp),%r13
  4010e3:   4c 89 ee                mov    %r13,%rsi
  4010e6:   e8 bf 04 00 00          callq  4015aa <read_six_numbers>
# 读取六个数字——用户应当输入六个数字
  4010eb:   4d 89 ee                mov    %r13,%r14
  4010ee:   41 bc 00 00 00 00       mov    $0x0,%r12d
# %r12d=0
  4010f4:   4c 89 ed                mov    %r13,%rbp
  4010f7:   41 8b 45 00             mov    0x0(%r13),%eax
  4010fb:   83 e8 01                sub    $0x1,%eax
  4010fe:   83 f8 05                cmp    $0x5,%eax
  401101:   76 05                   jbe    401108 <phase_6+0x36>
  401103:   e8 6c 04 00 00          callq  401574 <explode_bomb>
# 以上四句说明%eax 应当不大于 6
  401108:   41 83 c4 01             add    $0x1,%r12d
  40110c:   41 83 fc 06             cmp    $0x6,%r12d
  401110:   74 22                   je     401134 <phase_6+0x62>
  401112:   44 89 e3                mov    %r12d,%ebx
  401115:   48 63 c3                movslq %ebx,%rax
# %rax=%r12d 加 1 后算术扩展
  401118:   8b 44 84 30             mov    0x30(%rsp,%rax,4),%eax
# 取以 0x30+%rsp 开头的第 %rax 个数，放到 %eax 中
  40111c:   39 45 00                cmp    %eax,0x0(%rbp)
# 0x0(%rbp)=0x30(%rsp)
  40111f:   75 05                   jne    401126 <phase_6+0x54>
  401121:   e8 4e 04 00 00          callq  401574 <explode_bomb>
# 如果有重复数字则引发炸弹
  401126:   83 c3 01                add    $0x1,%ebx
  401129:   83 fb 05                cmp    $0x5,%ebx
  40112c:   7e e7                   jle    401115 <phase_6+0x43>
# 此处循环实现对数字重复的判断
  40112e:   49 83 c5 04             add    $0x4,%r13
# 通过%r13 加 4，实现上面的%eax 多读一个数
  401132:   eb c0                   jmp    4010f4 <phase_6+0x22>
# 循环。前面相当于要求输入的六个数互不相同，并且要小于等于 6
# 以上，第一部分内容结束，主要功能是读取六个数字并作预先判断

  401134:   48 8d 74 24 48          lea    0x48(%rsp),%rsi
# 跳转到这句话时，六个数都已读取并预判断结束。%rsi=0x48+%rsp（值）
  0x48-0x30=0x18=24，恰好是 6 个数字的空间
```

```
  401139:   4c 89 f0              mov    %r14,%rax
  40113c:   b9 07 00 00 00        mov    $0x7,%ecx
  401141:   89 ca                 mov    %ecx,%edx
  401143:   2b 10                 sub    (%rax),%edx
  401145:   89 10                 mov    %edx,(%rax)
# (%rax)=7-(%rax)
  401147:   48 83 c0 04           add    $0x4,%rax
# 取出下一个数
  40114b:   48 39 f0              cmp    %rsi,%rax
  40114e:   75 f1                 jne    401141 <phase_6+0x6f>
# 此处循环，即对输入的六个数进行处理：对每一个数，x=7-x
# 以上，第二部分内容结束

  401150:   be 00 00 00 00        mov    $0x0,%esi
  401155:   eb 20                 jmp    401177 <phase_6+0xa5>
  401157:   48 8b 52 08           mov    0x8(%rdx),%rdx
# %rbx=0x8+%rdx（值）
  40115b:   83 c0 01              add    $0x1,%eax
# %eax+=1
  40115e:   39 c8                 cmp    %ecx,%eax
  401160:   75 f5                 jne    401157 <phase_6+0x85>
  401162:   eb 05                 jmp    401169 <phase_6+0x97>
  401164:   ba f0 42 60 00        mov    $0x6042f0,%edx
# 出现了地址 0x6042f0，其值为
```

```
(gdb) x/24 0x6042f0
0x6042f0 <node1>:       880     1       6308608 0
0x604300 <node2>:       366     2       6308624 0
0x604310 <node3>:       526     3       6308640 0
0x604320 <node4>:       837     4       6308656 0
0x604330 <node5>:       777     5       6308672 0
0x604340 <node6>:       270     6       0       0
```

这是一个结构体，形式如下：

```
Struct node
{
    Int value;
    Int order;
    Node * next;
}
```

```
  401169:   48 89 14 74           mov    %rdx,(%rsp,%rsi,2)
# %rsp+%rsi*2 的值为结构体中的值
  40116d:   48 83 c6 04           add    $0x4,%rsi
  401171:   48 83 fe 18           cmp    $0x18,%rsi
  401175:   74 15                 je     40118c <phase_6+0xba>
# 把结构体拷贝到寄存器中
  401177:   8b 4c 34 30           mov    0x30(%rsp,%rsi,1),%ecx
```

```
# 第一次跳到该语句时，%ecx 为处理后数组的第一个数
  40117b:   83 f9 01                cmp    $0x1,%ecx
  40117e:   7e e4                   jle    401164 <phase_6+0x92>
  401180:   b8 01 00 00 00          mov    $0x1,%eax
  401185:   ba f0 42 60 00          mov    $0x6042f0,%edx
# 把结构体赋给%edx
  40118a:   eb cb                   jmp    401157 <phase_6+0x85>
  40118c:   48 8b 1c 24             mov    (%rsp),%rbx
  401190:   48 8d 44 24 08          lea    0x8(%rsp),%rax
  401195:   48 8d 74 24 30          lea    0x30(%rsp),%rsi
# 取出数组中的数
  40119a:   48 89 d9                mov    %rbx,%rcx
  40119d:   48 8b 10                mov    (%rax),%rdx
# 取出 node
  4011a0:   48 89 51 08             mov    %rdx,0x8(%rcx)
  4011a4:   48 83 c0 08             add    $0x8,%rax
  4011a8:   48 39 f0                cmp    %rsi,%rax
  4011ab:   74 05                   je     4011b2 <phase_6+0xe0>
  4011ad:   48 89 d1                mov    %rdx,%rcx
  4011b0:   eb eb                   jmp    40119d <phase_6+0xcb>
# 循环，为取出输入数字对应的结构体的值
  4011b2:   48 c7 42 08 00 00 00    movq   $0x0,0x8(%rdx)
  4011b9:   00
  4011ba:   bd 05 00 00 00          mov    $0x5,%ebp
  4011bf:   48 8b 43 08             mov    0x8(%rbx),%rax
# %rax(地址)=%rbx 存着的地址+0x8
  4011c3:   8b 00                   mov    (%rax),%eax
  4011c5:   39 03                   cmp    %eax,(%rbx)
# %rbx 地址中的值与 %eax 比较
  4011c7:   7d 05                   jge    4011ce <phase_6+0xfc>
# 比较结构体中两个 node 的 value，前一个数大于等于后一个数跳转。这句话决定了我们输入指令对结构体进行排序的依据。
  4011c9:   e8 a6 03 00 00          callq  401574 <explode_bomb>
  4011ce:   48 8b 5b 08             mov    0x8(%rbx),%rbx
  4011d2:   83 ed 01                sub    $0x1,%ebp
# 以上，第三部分内容为按照我们输入的数字与 7 的差比较结构体中 value 值的大小。

  4011d5:   75 e8                   jne    4011bf <phase_6+0xed>
  4011d7:   48 83 c4 50             add    $0x50,%rsp
  4011db:   5b                      pop    %rbx
  4011dc:   5d                      pop    %rbp
  4011dd:   41 5c                   pop    %r12
  4011df:   41 5d                   pop    %r13
  4011e1:   41 5e                   pop    %r14
```

```
  4011e3:   c3                      retq
```

函数功能：已知形式如上所述的结构体，用户输入六个数字 1-6，系统把每个数字与 7 的差作为 order，取出相应次序的结构体，如果取出结构体的 value 值从大到小排列则通过。

按照正常次序对结构体排序，得到 1 4 5 3 2 6，对应的输入为 6 3 2 4 5 1.

>>秘密关卡

在<phase_6>下面有一个名为<fun7>的函数，再往下就出现了隐藏关卡<secret_phase>。先研究怎么进入隐藏关卡：用 wps 的查找功能查找汇编代码，发现在<phase_defused>函数中出现了<secret_phase>的调用。

```
0000000000401712 <phase_defused>:
  401712:   48 83 ec 68             sub    $0x68,%rsp
  401716:   bf 01 00 00 00          mov    $0x1,%edi
  40171b:   e8 90 fd ff ff          callq  4014b0 <send_msg>
  401720:   83 3d 75 30 20 00 06    cmpl   $0x6,0x203075(%rip)  # 60479c
<num_input_strings>
  401727:   75 6d                   jne    401796 <phase_defused+0x84>
  401729:   4c 8d 44 24 10          lea    0x10(%rsp),%r8
  40172e:   48 8d 4c 24 08          lea    0x8(%rsp),%rcx
  401733:   48 8d 54 24 0c          lea    0xc(%rsp),%rdx
# 输入值
  401738:   be 37 28 40 00          mov    $0x402837,%esi
# 此处内容为
(gdb) x/s 0x402837
0x402837:       "%d %d %s"
  40173d:   bf b0 48 60 00          mov    $0x6048b0,%edi
  401742:   b8 00 00 00 00          mov    $0x0,%eax
  401747:   e8 e4 f4 ff ff          callq  400c30 <__isoc99_sscanf@plt>
  40174c:   83 f8 03                cmp    $0x3,%eax
# 又看到了 sscanf 函数。判断读进来的数是不是三个
  40174f:   75 31                   jne    401782 <phase_defused+0x70>
# 如果不是三个，说明没有发现隐藏关卡，跳到第 112 行，输出实验正常结束的
“Congratulations! You’ve defused the bomb! ”
  401751:   be 40 28 40 00          mov    $0x402840,%esi
# 此处内容为
(gdb) x/s 0x402840
0x402840:       "DrEvil"
  401756:   48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  40175b:   e8 ae fb ff ff          callq  40130e <strings_not_equal>
# 比较输入的最后一个值是否和“DrEvil”相对应
  401760:   85 c0                   test   %eax,%eax
```

```
  401762:   75 1e                  jne    401782 <phase_defused+0x70>
# 如果不对应，则也是正常结束实验
  401764:   bf 98 26 40 00         mov    $0x402698,%edi
# 此处内容为
```

```
(gdb) x/s 0x402698
0x402698:       "Curses, you've found the secret phase!"
```

```
  401769:   e8 d2 f3 ff ff         callq  400b40 <puts@plt>
  40176e:   bf c0 26 40 00         mov    $0x4026c0,%edi
# 此处内容为
```

```
(gdb) x/s 0x4026c0
0x4026c0:       "But finding it and solving it are quite different..."
```

```
  401773:   e8 c8 f3 ff ff         callq  400b40 <puts@plt>
  401778:   b8 00 00 00 00         mov    $0x0,%eax
  40177d:   e8 a0 fa ff ff         callq  401222 <secret_phase>
# 此处调用 secret_phase 函数。
  401782:   bf f8 26 40 00         mov    $0x4026f8,%edi
  401787:   e8 b4 f3 ff ff         callq  400b40 <puts@plt>
  40178c:   bf 28 27 40 00         mov    $0x402728,%edi
# 实验正常结束时打出来的字
```

```
(gdb) x/s 0x4026f8
0x4026f8:       "Congratulations! You've defused the bomb!"
(gdb) x/s 0x402728
0x402728:       "Your instructor has been notified and will verify your solution
."
```

```
  401791:   e8 aa f3 ff ff         callq  400b40 <puts@plt>
  401796:   48 83 c4 68            add    $0x68,%rsp
  40179a:   c3                     retq
  40179b:   0f 1f 44 00 00         nopl   0x0(%rax,%rax,1)
```

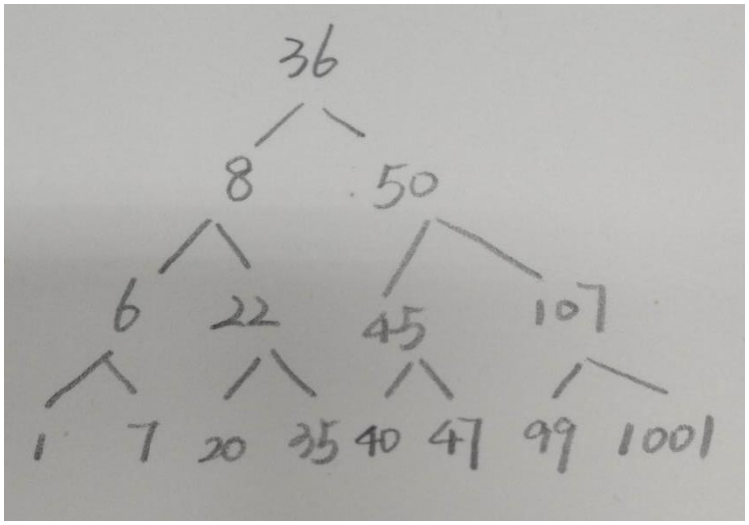
可以推断，我们要输入一个格式为“%d %d %s”的值，且最后一个字符串为“DrEvil”才能进入隐藏关卡。之前有类似格式的有第 4 题，所以试着在做第 4 题时输入“108 2 DrEvil”，发现就能进入隐藏关卡了。

```
0000000000401222 <secret_phase>:
  401222:   53                     push   %rbx
  401223:   e8 c4 03 00 00         callq  4015ec <read_line>
# 读行
  401228:   ba 0a 00 00 00         mov    $0xa,%edx
  40122d:   be 00 00 00 00         mov    $0x0,%esi
  401232:   48 89 c7               mov    %rax,%rdi
  401235:   e8 c6 f9 ff ff         callq  400c00 <strtol@plt>
# strtol 函数会将参数 nptr 字符串根据参数 base 来转换成长整型数
  40123a:   48 89 c3               mov    %rax,%rbx
  40123d:   8d 40 ff               lea    -0x1(%rax),%eax
  401240:   3d e8 03 00 00         cmp    $0x3e8,%eax
```

```
# 把结果和 1000 进行比较
  401245:   76 05                   jbe    40124c <secret_phase+0x2a>
# 比 1000 小的话，则执行 fun7
  401247:   e8 28 03 00 00          callq  401574 <explode_bomb>
  40124c:   89 de                   mov    %ebx,%esi
# 一个参数
  40124e:   bf 10 41 60 00          mov    $0x604110,%edi
# 另一个参数，其值为
```

```
(gdb) x/120 0x604110
0x604110 <n1>:  36      0       6308144 0
0x604120 <n1+16>:       6308176 0       0       0
0x604130 <n21>: 8       0       6308272 0
0x604140 <n21+16>:      6308208 0       0       0
0x604150 <n22>: 50      0       6308240 0
0x604160 <n22+16>:      6308304 0       0       0
0x604170 <n32>: 22      0       6308496 0
0x604180 <n32+16>:      6308432 0       0       0
0x604190 <n33>: 45      0       6308336 0
0x6041a0 <n33+16>:      6308528 0       0       0
0x6041b0 <n31>: 6       0       6308368 0
0x6041c0 <n31+16>:      6308464 0       0       0
0x6041d0 <n34>: 107     0       6308400 0
0x6041e0 <n34+16>:      6308560 0       0       0
0x6041f0 <n45>: 40      0       0       0
0x604200 <n45+16>:      0       0       0       0
0x604210 <n41>: 1       0       0       0
0x604220 <n41+16>:      0       0       0       0
0x604230 <n47>: 99      0       0       0
0x604240 <n47+16>:      0       0       0       0
0x604250 <n44>: 35      0       0       0
0x604260 <n44+16>:      0       0       0       0
0x604270 <n42>: 7       0       0       0
---Type <return> to continue, or q <return> to quit---
0x604280 <n42+16>:      0       0       0       0
0x604290 <n43>: 20      0       0       0
0x6042a0 <n43+16>:      0       0       0       0
0x6042b0 <n46>: 47      0       0       0
0x6042c0 <n46+16>:      0       0       0       0
0x6042d0 <n48>: 1001    0       0       0
0x6042e0 <n48+16>:      0       0       0       0
```

可以看出采用了孩子表示法，画出来就是这样的树：




```
  401253:   e8 8c ff ff ff          callq  4011e4 <fun7>
  401258:   83 f8 06                cmp    $0x6,%eax
# 返回值需要是 6
  40125b:   74 05                   je     401262 <secret_phase+0x40>
  40125d:   e8 12 03 00 00          callq  401574 <explode_bomb>
  401262:   bf f8 24 40 00          mov    $0x4024f8,%edi
  401267:   e8 d4 f8 ff ff          callq  400b40 <puts@plt>
  40126c:   e8 a1 04 00 00          callq  401712 <phase_defused>
  401271:   5b                      pop    %rbx
  401272:   c3                      retq
  401273:   66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
  40127a:   00 00 00
  40127d:   0f 1f 00                nopl   (%rax)
```

可以看出，隐藏关卡需要我们输入一个小于一千的数，传到 fun7 里，使它的返回值为 6。然后分析 fun7 函数。

```
00000000004011e4 <fun7>:
  4011e4:   48 83 ec 08             sub    $0x8,%rsp
  4011e8:   48 85 ff                test   %rdi,%rdi
  4011eb:   74 2b                   je     401218 <fun7+0x34>
# 如果第一个参数（根据后面分析是空指针）为 0，则返回值是-1
  4011ed:   8b 17                   mov    (%rdi),%edx
  4011ef:   39 f2                   cmp    %esi,%edx
# 比较第二个参数和树结点的值，如果树节点的值小于等于第二个参数则语句跳转
  4011f1:   7e 0d                   jle    401200 <fun7+0x1c>
  4011f3:   48 8b 7f 08             mov    0x8(%rdi),%rdi
# 如果大于，则看左孩子结点
  4011f7:   e8 e8 ff ff ff          callq  4011e4 <fun7>
  4011fc:   01 c0                   add    %eax,%eax
# 返回值乘 2
```

```
  4011fe:   eb 1d                   jmp    40121d <fun7+0x39>
  401200:   b8 00 00 00 00          mov    $0x0,%eax
  401205:   39 f2                   cmp    %esi,%edx
  401207:   74 14                   je     40121d <fun7+0x39>
  401209:   48 8b 7f 10             mov    0x10(%rdi),%rdi
# 如果不相等，则传入右孩子结点；如果相等，则返回 0
  40120d:   e8 d2 ff ff ff          callq  4011e4 <fun7>
# 递归调用
  401212:   8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
# 返回值乘 2 加 1
  401216:   eb 05                   jmp    40121d <fun7+0x39>
  401218:   b8 ff ff ff ff          mov    $0xffffffff,%eax
  40121d:   48 83 c4 08             add    $0x8,%rsp
  401221:   c3                      retq
```

写成 c 语言函数，就是这样的形式：

```
struct Node
{
    int data;
    struct Node* leftChild;
    struct Node* rightChild;
};

int fun7(struct Node* p, int x)
{
    if (p == NULL)
        return -1;
    else if (x == p->data)
        return 0;
    else if (x < p->data)
        return 2 × fun7(p->leftChild, x);
    else
        return 2 × fun7(p->rightChild, x) + 1;
}
```

要让 fun7 返回值为 6，对照上面的二叉树图和递归函数可以知道应当输入 35.

五、个人总结

首先，通过 Bomblab 实验，我最大的收获是阅读和理解汇编语言的能力得到了锻炼。此前面对大篇幅的汇编语句会有无从下手、不知所措的感觉，在这次实验之后，这种现象有了改善。拿到量大的代码可以逐条阅读理解下去，哪怕不能理解细节的实现，也能对函数功能有个大概的印象。

其次，对实验中的汇编语句，有些第一眼看上去会觉得摸不着头脑；在课上

讲了 control 和 procedure 之后，我才慢慢理解它为什么产生这样的汇编语句（比如 push %ebp 这样的操作），做实验对上课讲授的知识有较好的巩固作用。

就解题过程而言，在整个 Bomblab 实验中起关键作用的往往是比较和跳转，需要对此类语句加以特别注意。另外，根据%esp 的加减判断指向哪一块内存，也是后面解题时容易混淆的地方。而像把二叉树、结构体放进汇编里的用法，在平时课堂上没怎么看到，也算是通过实验增长了见识。