CSAPP-BOMB-LAB

PB16030899-朱河勤
2018-3-16

下载得到bomb.tar文件,解压后只有bomb二进制文件,以及一个bomb.c文件,bomb.c没有对应的头文件. 所有思路只有是反汇编bomb,分析汇编代码.

这里用到两个非常强大的工具objdump,gdb

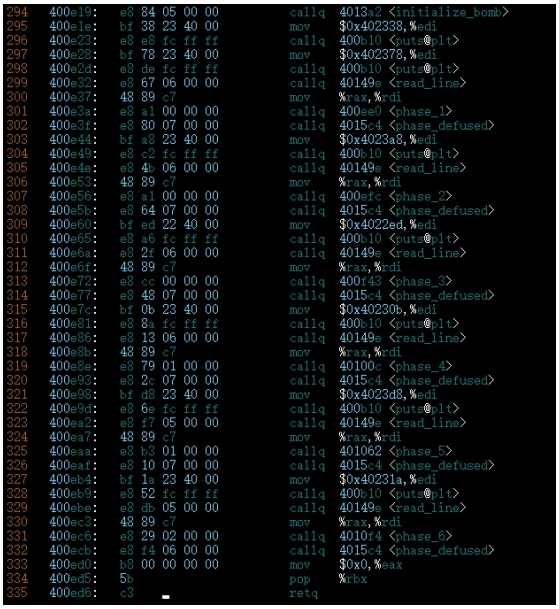
- objdump用来反汇编的,-d参数得到x86汇编, -M参数还可以选择不同的汇编形式, 比如 -M 8086 得到8086汇编, 详细内容可以man objdump.
- gdb是强大的GNU DEBUGGER 用法如下

```
(gdb) b（breakpoint）:用法：b 函数名 ：对此函数进行中断 ；b 文件名：行号；
(gdb) run：启动程序，运行至程序的断点或者结束；
(gdb) l(list):用法：l funcname，制定函数的源码
(gdb) s(step):进入函数，逐语句运行；
(gdb) n(next):不进入函数，逐过程运行；
(gdb) c（continue）：继续运行，跳至下一个断点；
(gdb) p（print）：打印显示变量值；
(gdb) set variable=value,为变量赋值；
(gdb) kill：终止调试的程序；
(gdb) h（help）：列出gdb详细命令帮助列表；
(gdb) clear filename.c:30：清除30行处的断点；
(gdb) info break：显示断点信息；
(gdb) delete 断点编号：断点编号是info break 后显示出来的；
(gdb) bt（backtrace）：回溯到段出错的位置；
(gdb) frame 帧号：帧号是bt命令产生的堆栈针；
(gdb) q：退出；
(gdb) x(examine)：查看内存中的值等//详细内容在gdb中输入 help x查看
```

下面开始拆  之旅

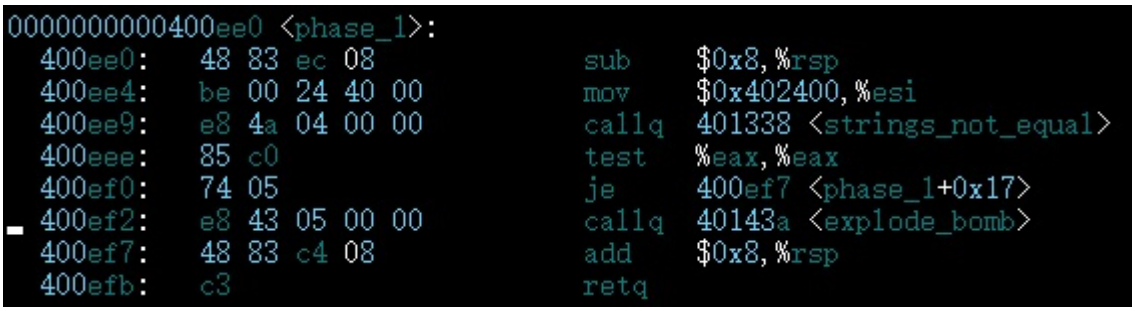
general

观察汇编代码,可以看到有main, phase1--6, 等, 重点看这几个函数, 从main开始, 结合bomb.c,可以明白程序的控制流, 每个阶段用phase函数判断输入是否正确,不正确就boon,结束程序



phase1

来到phase1,



第一行准备栈帧,第二行就是将地址存入$esi, 这是一个字符串的地址, 可以猜测下面string_not_equal就是比较这个字符串与输入字符串是否相等的函数.(最开始我还去分析了这个函数的汇编代码,确实是那样,先比较长度,然后逐一比较. 所以找到这个地址`0x402400`存储的字符串就行了,在asm文件中搜索,没有,所以要在程序运行时才可以到达这个虚拟地址, 未来address space 的堆中. 这时就要用到强大的gdb了,

切换到bomb文件夹,依次输入

```
gdb
(gdb) file bomb
(gdb) x /s 0x402400 # x(examine) s参数是string的意思
```

即得**Border relations with Canada have never been better.**

phase2

```
0000000000400efc <phase_2>:
  400efc:   55                      push   %rbp
  400efd:   53                      push   %rbx
  400efe:   48 83 ec 28             sub    $0x28,%rsp
  400f02:   48 89 e6                mov    %rsp,%rsi
  400f05:   e8 52 05 00 00          callq  40145c <read_six_numbers>     调用函数读取6个数字存于栈帧中(rsp)
  400f0a:   83 3c 24 01             cmpl   $0x1,(%rsp)                  第一个数为
  400f0e:   74 20                   je     400f30 <phase_2+0x34>
  400f10:   e8 25 05 00 00          callq  40143a <explode_bomb>
  400f15:   eb 19                   jmp    400f30 <phase_2+0x34>
  400f17:   8b 43 fc                mov    -0x4(%rbx),%eax
  400f1a:   01 c0                   add    %eax,%eax                     后面的数是
  400f1c:   39 03                   cmp    %eax,(%rbx)                   前面的数的
  400f1e:   74 05                   je     400f25 <phase_2+0x29>         两倍
  400f20:   e8 15 05 00 00          callq  40143a <explode_bomb>
  400f25:   48 83 c3 04             add    $0x4,%rbx
  400f29:   48 39 eb                cmp    %rbp,%rbx
  400f2c:   75 e9                   jne    400f17 <phase_2+0x1b>
  400f2e:   eb 0c                   jmp    400f3c <phase_2+0x40>
  400f30:   48 8d 5c 24 04          lea    0x4(%rsp),%rbx
  400f35:   48 8d 6c 24 18          lea    0x18(%rsp),%rbp
  400f3a:   eb db                   jmp    400f17 <phase_2+0x1b>
  400f3c:   48 83 c4 28             add    $0x28,%rsp
  400f40:   5b                      pop    %rbx
  400f41:   5d                      pop    %rbp
  400f42:   c3                      retq
```

所以答案是 1 2 4 8 16 32

phase3

```
  0000000000400f43 <phase_3>:
    400f43: sub    $0x18,%rsp
    400f47: lea    0xc(%rsp),%rcx
    400f4c: lea    0x8(%rsp),%rdx
    400f51: mov    $0x4025cf,%esi                  # 又是一个字符串,可以用gdb
查看, 得到`"%d %d",格式化字符串,说明输入两个数字
    400f56: mov    $0x0,%eax
    400f5b: callq  400bf0 <__isoc99_sscanf@plt>    # 输入
    400f60: cmp    $0x1,%eax                        # 判断输入成功
    400f63: jg     400f6a <phase_3+0x27>
    400f65: callq  40143a <explode_bomb>
    400f6a: cmpl   $0x7,0x8(%rsp)                     # 第一个参数是否小于等于
7,大于则boom
    400f6f: ja     400fad <phase_3+0x6a>
    400f71: mov    0x8(%rsp),%eax
    400f75: jmpq   *0x402470(,%rax,8)              # 以下是switch, 根据rax,即
第一个输入的参数跳转
    400f7c: mov    $0xcf,%eax                       # 由此容易得到答案, 比如这
里是rax=0时, 则 另一个参数为0xcf  = 207
    400f81: jmp    400fbe <phase_3+0x7b>
    400f83: mov    $0x2c3,%eax
    400f88: jmp    400fbe <phase_3+0x7b>
    400f8a: mov    $0x100,%eax
    400f8f: jmp    400fbe <phase_3+0x7b>
    400f91: mov    $0x185,%eax
    400f96: jmp    400fbe <phase_3+0x7b>
    400f98: mov    $0xce,%eax
    400f9d: jmp    400fbe <phase_3+0x7b>
    400f9f: mov    $0x2aa,%eax
    400fa4: jmp    400fbe <phase_3+0x7b>
```

```
    400fa6: mov    $0x147,%eax
    400fab: jmp    400fbe <phase_3+0x7b>
    400fad: callq  40143a <explode_bomb>
    400fb2: mov    $0x0,%eax
    400fb7: jmp    400fbe <phase_3+0x7b>
    400fb9: mov    $0x137,%eax
    400fbe: cmp    0xc(%rsp),%eax
    400fc2: je     400fc9 <phase_3+0x86>
    400fc4: callq  40143a <explode_bomb>
    400fc9: add    $0x18,%rsp
    400fcd: retq
```

swith跳转表 %rax 跳转地址 0xc(%rsp) 0 0x0000000000400f7c 0xcf 207 1 0x0000000000400fb9 0x137 311 2 0x0000000000400f83 0x2c3 707 3 0x0000000000400f8a 0x100 256 4 0x0000000000400f91 0x185 389 5 0x0000000000400f98 0xce 206 6 0x0000000000400f9f 0x2aa 682 7 0x0000000000400fa6 0x147 327 所以结果为`0 207` ...

phase4

```
 000000000040100c <phase_4>:
   40100c:    sub    $0x18,%rsp
   401010:    lea    0xc(%rsp),%rcx
   401015:    lea    0x8(%rsp),%rdx
   40101a:    mov    $0x4025cf,%esi                  #同样,gdb 中x /s 知道输入两个
数字
   40101f:    mov    $0x0,%eax
   401024:    callq  400bf0 <__isoc99_sscanf@plt>
   401029:    cmp    $0x2,%eax                       # 判断是否输入两个数
   40102c:    jne    401035 <phase_4+0x29>
   40102e:    cmpl   $0xe,0x8(%rsp)                   # 判断每个数是否≤14 ,大于则
boom
   401033:    jbe    40103a <phase_4+0x2e>            # 跳转
   401035:    callq  40143a <explode_bomb>
   40103a:    mov    $0xe,%edx                        # 构造func4的参数 (phase4调
用的)
   40103f:    mov    $0x0,%esi                        # 构造func4的参数
   401044:    mov    0x8(%rsp),%edi                    # 构造func4的参数
   401048:    callq  400fce <func4>
   40104d:    test   %eax,%eax                        # 测试, func4返回0, 若不,则
boom
   40104f:    jne    401058 <phase_4+0x4c>
   401051:    cmpl   $0x0,0xc(%rsp)
   401056:    je     40105d <phase_4+0x51>
   401058:    callq  40143a <explode_bomb>
   40105d:    add    $0x18,%rsp
   401061:    retq
```

将func4转换为c语言,并用0--14测试, 这点很难, 需要翻译汇编语言,花很多时间,得熟悉汇编代码才行

```
int func4(int a, int b, int c)
{
   int result;
   result = c;
   result = result - b;
   int tmp = result;
   tmp = (unsigned)tmp >> 31;
   result = result + tmp;
   result = result / 2;
   tmp = result + b;
   if(tmp > a)
   {
       c = tmp - 1;
       result = func4(a, b, c);
       return (2 * result);
   }
   result = 0;
   if(tmp < a)
   {
       b = tmp + 1;
       result = func4(a, b, c);
       return (1 + 2 * result);
   }
   return result;
}
//测试从0~14范围内满足条件的值

int main()
{
   for(int input = 0; input < 15; ++input)
   {
       int result = func4(input, 0, 14);
       if(result == 0)
       {
           printf("input = %d, func4 = %d\n", input, result);
       }
   }
   return 0;
}
```

得到可行解 因此phase4可能结果为： 0 0 1 0 3 0 7 0

phase5

嗯, 加油, 还有两关了. (●ˇ∀ˇ●)

```
 0000000000401062 <phase_5>:
```

```
    401062:  push   %rbx
    401063:  sub    $0x20,%rsp
    401067:  mov    %rdi,%rbx
    40106a:  mov    %fs:0x28,%rax
    401071:
    401073:  mov    %rax,0x18(%rsp)
    401078:  xor    %eax,%eax
    40107a:  callq  40131b <string_length>
    40107f:  cmp    $0x6,%eax                  # 说明输入是六个字符
    401082:  je     4010d2 <phase_5+0x70>
    401084:  callq  40143a <explode_bomb>
    401089:  jmp    4010d2 <phase_5+0x70>
    40108b:  movzbl (%rbx,%rax,1),%ecx        # 从栈帧中取出各个字符,记为x
    40108f:  mov    %cl,(%rsp)
    401092:  mov    (%rsp),%rdx
    401096:  and    $0xf,%edx                 # y=0xf & x, 即将一个byte的高4位置0
    401099:  movzbl 0x4024b0(%rdx),%edx        # 用gdb查看x /s 0x4024b0  得到字符
串"maduiersnfotvbyl",所以这一行是以y作为偏移量,取字符数组的第几个字符
    4010a0:  mov    %dl,0x10(%rsp,%rax,1)      # 将取得的存于栈帧中 //后面用
string_not_equl 比较
    4010a4:  add    $0x1,%rax
    4010a8:  cmp    $0x6,%rax                  # 循环6次
    4010ac:  jne    40108b <phase_5+0x29>
    4010ae:  movb   $0x0,0x16(%rsp)
    4010b3:  mov    $0x40245e,%esi            # 这是要比较的字符串, 同样用gdb查看得
到 "flyers"
    4010b8:  lea    0x10(%rsp),%rdi
    4010bd:  callq  401338 <strings_not_equal>
    4010c2:  test   %eax,%eax
    4010c4:  je     4010d9 <phase_5+0x77>
    4010c6:  callq  40143a <explode_bomb>
    4010cb:  nopl   0x0(%rax,%rax,1)
    4010d0:  jmp    4010d9 <phase_5+0x77>
    4010d2:  mov    $0x0,%eax
    4010d7:  jmp    40108b <phase_5+0x29>
    4010d9:  mov    0x18(%rsp),%rax
    4010de:  xor    %fs:0x28,%rax
    4010e5:
    4010e7:  je     4010ee <phase_5+0x8c>
    4010e9:  callq  400b30 <__stack_chk_fail@plt>
    4010ee:  add    $0x20,%rsp
    4010f2:  pop    %rbx
    4010f3:  retq
```

解释在上面, 反向得到需要的输入的思路是: 对flyers的每个字符, 得到在字符数组中的index, 也就是输入的字符的后4位bit, 而键盘输入一般是字母, 所以很可能有两种可能,字符byte的高四位为`0100`或`0110`,而且可以发现刚好这是大写字母/小写字母开始的前一个ascii,所以

```
>>> li=list('maduiersnfotvbyl')
>>> li
['m', 'a', 'd', 'u', 'i', 'e', 'r', 's', 'n', 'f', 'o', 't', 'v', 'b', 'y',
 'l']
>>> ''.join([chr(li.index(i)+(1<<6)) for i in "flyers"])
'IONEFG'
>>> ''.join([chr(li.index(i)+((1<<6)+(1<<7))) for i in "flyers"])
'ÉÏÎÅÆÇ'
>>> ''.join([chr(li.index(i)+((1<<6)+(1<<5))) for i in "flyers"])
'ionefg'
>>>
```

phase6

phase6很难了,这真的要熟练汇编语言, 翻译一下,知道输入的是六个不相同的数字, 而且≤6 ,~~所以可以试全排列子~~

```
(gdb) disas phase_6
Dump of assembler code for function phase_6:
  0x00000000004010f4 <+0>: push   %r14                                    将被调用者
保存寄存器压入栈
  0x00000000004010f6 <+2>: push   %r13
  0x00000000004010f8 <+4>: push   %r12
  0x00000000004010fa <+6>: push   %rbp
  0x00000000004010fb <+7>: push   %rbx                                    %rsp =
0x7fffffffe2c0
  0x00000000004010fc <+8>: sub    $0x50,%rsp                              分配栈空间
%rsp = 0x7fffffffe270
  0x0000000000401100 <+12>:    mov    %rsp,%r13

  0x0000000000401103 <+15>:    mov    %rsp,%rsi
  0x0000000000401106 <+18>:    callq  0x40145c <read_six_numbers>      读入6个
值,保存至从 %rsi 开始的地址

  0x000000000040110b <+23>:    mov    %rsp,%r14
  0x000000000040110e <+26>:    mov    $0x0,%r12d                       %r12 置0,
并且%r13 %r14 %rbp 均和 %rsp 指向相同地址 0x7fffffffe270

  0x0000000000401114 <+32>:    mov    %r13,%rbp
  0x0000000000401117 <+35>:    mov    0x0(%r13),%eax                   将第 %r13
指向的输入数复制到 %eax
  0x000000000040111b <+39>:    sub    $0x1,%eax                        将输入数减
1
  0x000000000040111e <+42>:    cmp    $0x5,%eax                        判断输入数
是否小于等于6,因为上一步中减1操作
  0x0000000000401121 <+45>:    jbe    0x401128 <phase_6+52>            若大于6,
则调用 explode_bomb
  0x0000000000401123 <+47>:    callq  0x40143a <explode_bomb>
================================================================================
================================================================================
  0x0000000000401128 <+52>:    add    $0x1,%r12d                       将 %r12
加1
  0x000000000040112c <+56>:    cmp    $0x6,%r12d                       判断 %r12
```

```
是否等于6
  0x0000000000401130 <+60>:    je     0x401153 <phase_6+95>           若等于6,
跳转,否则继续执行
  0x0000000000401132 <+62>:    mov    %r12d,%ebx                      将 %r12
复制到 %ebx

  0x0000000000401135 <+65>:    movslq %ebx,%rax                       将 %ebx
符号位扩展复制到 %rax
  0x0000000000401138 <+68>:    mov    (%rsp,%rax,4),%eax              将第 %ebx
输入数复制到 %eax
  0x000000000040113b <+71>:    cmp    %eax,0x0(%rbp)                  比较 %r13
指向的输入数和 第 %ebx 输入数 是否相等
  0x000000000040113e <+74>:    jne    0x401145 <phase_6+81>           如果相等,
则调用 explode_bomb
  0x0000000000401140 <+76>:    callq  0x40143a <explode_bomb>
  0x0000000000401145 <+81>:    add    $0x1,%ebx                       将 %ebx
加1
  0x0000000000401148 <+84>:    cmp    $0x5,%ebx                       判断 %ebx
是否小于等于5
  0x000000000040114b <+87>:    jle    0x401135 <phase_6+65>           若小于等
于,跳转,否则继续执行;该循环判断 %r13 指向的数据和其后输入数不相等

  0x000000000040114d <+89>:    add    $0x4,%r13                       将 %r13
指向下一个输入数,该循环判断所有的输入数全部不相等
  0x0000000000401151 <+93>:    jmp    0x401114 <phase_6+32>
================================================================================
================================================================================
  0x0000000000401153 <+95>:    lea    0x18(%rsp),%rsi                 将 %rsi
指向栈中跳过读入数据位置作为结束标记,并且 %r14 仍和 %rsp 指向同一个位置
  0x0000000000401158 <+100>:   mov    %r14,%rax                       将 %r14
复制到 %rax
  0x000000000040115b <+103>:   mov    $0x7,%ecx
  0x0000000000401160 <+108>:   mov    %ecx,%edx                       将立即数
0x7复制到 %edx
  0x0000000000401162 <+110>:   sub    (%rax),%edx                     立即数7减
去 %r14 指向的数据
  0x0000000000401164 <+112>:   mov    %edx,(%rax)                     将7减的结
果存回 %r14 执行的内存单元
  0x0000000000401166 <+114>:   add    $0x4,%rax                       %rax 指向
下一个输入数
  0x000000000040116a <+118>:   cmp    %rsi,%rax                       比较是否达
到输入数组的末尾,
  0x000000000040116d <+121>:   jne    0x401160 <phase_6+108>          该循环使用
立即数7减去每个输入数据
================================================================================
================================================================================
  0x000000000040116f <+123>:   mov    $0x0,%esi                       将 %rsi
置0
  0x0000000000401174 <+128>:   jmp    0x401197 <phase_6+163>

  0x0000000000401176 <+130>:   mov    0x8(%rdx),%rdx                  将
0x8(%rdx) 指向内存单元的内容复制到 %rdx, 指向链表下一个元素
```

```
  0x000000000040117a <+134>:   add    $0x1,%eax                          将 %eax
加1
  0x000000000040117d <+137>:   cmp    %ecx,%eax                          比较 %ecx
和 %eax 是否相等
  0x000000000040117f <+139>:   jne    0x401176 <phase_6+130>             不相等,继
续遍历链表,最终 %rdx 指向链表的第 %ecx 个节点
  0x0000000000401181 <+141>:   jmp    0x401188 <phase_6+148>
  0x0000000000401183 <+143>:   mov    $0x6032d0,%edx                     重置链表首
地址
  0x0000000000401188 <+148>:   mov    %rdx,0x20(%rsp,%rsi,2)
  0x000000000040118d <+153>:   add    $0x4,%rsi
  0x0000000000401191 <+157>:   cmp    $0x18,%rsi
  0x0000000000401195 <+161>:   je     0x4011ab <phase_6+183>

  0x0000000000401197 <+163>:   mov    (%rsp,%rsi,1),%ecx                 将 (%rsp
+ %rsi) 指向的数据复制到 %ecx
  0x000000000040119a <+166>:   cmp    $0x1,%ecx                          比较 %ecx
是否小于等于1
  0x000000000040119d <+169>:   jle    0x401183 <phase_6+143>             若小于等
于,跳转,否则继续执行, 等于1, %edx 直接指向链表首地址
  0x000000000040119f <+171>:   mov    $0x1,%eax                          将 %eax
置1
  0x00000000004011a4 <+176>:   mov    $0x6032d0,%edx                     将 %rdx
指向内存单元 0x6032d0
  0x00000000004011a9 <+181>:   jmp    0x401176 <phase_6+130>             跳转; 该
循环根据输入数将链表中对应的第输入数个节点的地址复制到 0x20(%rsp) 开始的栈中
================================================================================
================================================================================
  0x00000000004011ab <+183>:   mov    0x20(%rsp),%rbx                    将
0x20(%rsp)的链表节点地址复制到 %rbx
  0x00000000004011b0 <+188>:   lea    0x28(%rsp),%rax                    将 %rax
指向栈中下一个链表节点的地址
  0x00000000004011b5 <+193>:   lea    0x50(%rsp),%rsi                    将 %rsi
指向保存的链表节点地址的末尾
  0x00000000004011ba <+198>:   mov    %rbx,%rcx

  0x00000000004011bd <+201>:   mov    (%rax),%rdx
  0x00000000004011c0 <+204>:   mov    %rdx,0x8(%rcx)                     将栈中指向
的后一个节点的地址复制到前一个节点的地址位置
  0x00000000004011c4 <+208>:   add    $0x8,%rax                          移动到下一
个节点
  0x00000000004011c8 <+212>:   cmp    %rsi,%rax                          判断6个节
点是否遍历完毕
  0x00000000004011cb <+215>:   je     0x4011d2 <phase_6+222>
  0x00000000004011cd <+217>:   mov    %rdx,%rcx
  0x00000000004011d0 <+220>:   jmp    0x4011bd <phase_6+201>
  0x00000000004011d2 <+222>:   movq   $0x0,0x8(%rdx)                     该循环按照
7减去输入数据的索引重新调整链表
================================================================================
================================================================================
  0x00000000004011da <+230>:   mov    $0x5,%ebp
  0x00000000004011df <+235>:   mov    0x8(%rbx),%rax                     将 %rax
```

```
指向 %rbx 下一个链表节点
  0x00000000004011e3 <+239>:   mov    (%rax),%eax
  0x00000000004011e5 <+241>:   cmp    %eax,(%rbx)                    比较链表节
点中第一个字段值的大小,如果前一个节点值大于后一个节点值,跳转
  0x00000000004011e7 <+243>:   jge    0x4011ee <phase_6+250>
  0x00000000004011e9 <+245>:   callq  0x40143a <explode_bomb>
  0x00000000004011ee <+250>:   mov    0x8(%rbx),%rbx                 将 %rbx
向后移动,指向栈中下一个链表节点的地址
  0x00000000004011f2 <+254>:   sub    $0x1,%ebp                      判断循环是
否结束,该循环判断栈中重新调整后的链表节点是否按照降序排列
  0x00000000004011f5 <+257>:   jne    0x4011df <phase_6+235>
  0x00000000004011f7 <+259>:   add    $0x50,%rsp
  0x00000000004011fb <+263>:   pop    %rbx
  0x00000000004011fc <+264>:   pop    %rbp
  0x00000000004011fd <+265>:   pop    %r12
  0x00000000004011ff <+267>:   pop    %r13
  0x0000000000401201 <+269>:   pop    %r14
  0x0000000000401203 <+271>:   retq
End of assembler dump.

(gdb) disas read_six_numbers
%rsi存储调用者phase_2栈帧的局部变量开始地址
%rdx = %rsi + 0
%rcx = %rsi + 4
%r8 =  %rsi + 8
%r9 =  %rsi + 12
(%rsp)  = %rsi + 16
8(%rsp) = %rsi + 20
Dump of assembler code for function read_six_numbers:
  0x000000000040145c <+0>: sub    $0x18,%rsp
  0x0000000000401460 <+4>: mov    %rsi,%rdx
  0x0000000000401463 <+7>: lea    0x4(%rsi),%rcx
  0x0000000000401467 <+11>:    lea    0x14(%rsi),%rax
  0x000000000040146b <+15>:    mov    %rax,0x8(%rsp)
  0x0000000000401470 <+20>:    lea    0x10(%rsi),%rax
  0x0000000000401474 <+24>:    mov    %rax,(%rsp)
  0x0000000000401478 <+28>:    lea    0xc(%rsi),%r9
  0x000000000040147c <+32>:    lea    0x8(%rsi),%r8
  0x0000000000401480 <+36>:    mov    $0x4025c3,%esi
  0x0000000000401485 <+41>:    mov    $0x0,%eax
  0x000000000040148a <+46>:    callq  0x400bf0 <__isoc99_sscanf@plt>
  0x000000000040148f <+51>:    cmp    $0x5,%eax
  0x0000000000401492 <+54>:    jg     0x401499 <read_six_numbers+61>
  0x0000000000401494 <+56>:    callq  0x40143a <explode_bomb>
  0x0000000000401499 <+61>:    add    $0x18,%rsp
  0x000000000040149d <+65>:    retq
```

%rbp %rbx %r12~%15 被调用者保存寄存器 %r10 %r11 调用者保存寄存器 %rdi %rsi %rdx %rcx %r8 %r9 依次保存输入数1~6

假设输入数据为4 3 2 1 6 5

猜测0x6032d8为链表首地址,链表中每个节点占用12个Byte,前8字节保存两个4字Byte的整型数,剩余的4Byte存放下个节点地址

GDB查看使用7减去对应的输入后的数据 (gdb) p /x $rsp $1 = 0x7fffffffe270 (gdb) x/6dw 0x7fffffffe270 0x7fffffffe270: 3 4 5 6 0x7fffffffe280: 1 2

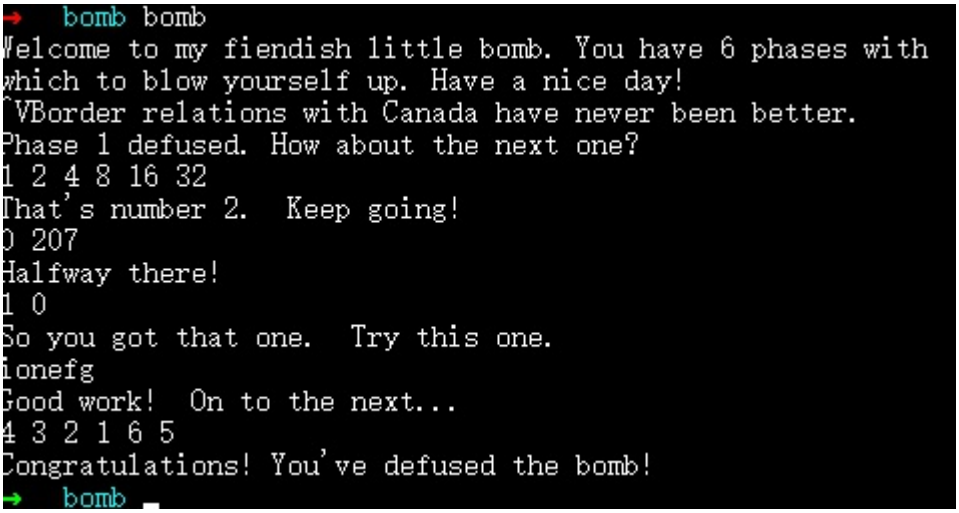
重新调整链表前的链表的结构 (gdb) x/24xw 0x006032d0 0x6032d0 : 0x0000014c 0x00000001 0x006032e0 0x00000000 0x6032e0 : 0x000000a8 0x00000002 0x006032f0 0x00000000 0x6032f0 : 0x0000039c 0x00000003 0x00603300 0x00000000 0x603300 : 0x000002b3 0x00000004 0x00603310 0x00000000 0x603310 : 0x000001dd 0x00000005 0x00603320 0x00000000 0x603320 : 0x000001bb 0x00000006 0x00000000 0x00000000

保存在栈中链表节点信息 (gdb) x/6xg 0x7fffffffe290 0x7fffffffe290: 0x00000000006032f0 0x0000000000603300 0x7fffffffe2a0: 0x0000000000603310 0x0000000000603320 0x7fffffffe2b0: 0x00000000006032d0 0x00000000006032e0

按照7减去对应的输入后重新调整链表后的链表结构,索引顺序为 3 4 5 6 1 2 (gdb) x/24xw 0x006032d0 0x6032d0 : 0x0000014c 0x00000001 0x006032e0 0x00000000 0x6032e0 : 0x000000a8 0x00000002 0x00000000 0x00000000 0x6032f0 : 0x0000039c 0x00000003 0x00603300 0x00000000 0x603300 : 0x000002b3 0x00000004 0x00603310 0x00000000 0x603310 : 0x000001dd 0x00000005 0x00603320 0x00000000 0x603320 : 0x000001bb 0x00000006 0x006032d0 0x00000000

破解思路: 将链表中每个节点按照前4字节降序排序 3 4 5 6 1 2 因为在前面使用7减去对应的值,所以破解密码 4 3 2 1 6 5

final



啊,终于拆除💣了, ╰(°▽°)╯ 等等, 还漏了什么? 在asm中,可以看到还有secret_phase这个函数,可是这个函数的调用是有技巧的,追踪发现是在phase_defused中调用的,同样, 查看字符串, 发现比较了"DrEvil", 以及一个格式串"%d %d %s",可能是phase3,phase4的数字加上DrEvil输入.可是最后我试了很多篇都没有是出来. 最后在gdb中设置断点, 然后jump secret_phase` 即可进入

总结

通过这个lab,学到了gdb,objdump等工具的使用, 对汇编语言更熟悉, 对函数调用中栈帧的变化, 动态变量的理解

更加深刻 不得不佩服作者,以及这个lab的有趣与实用