Bomblab Report

2021201709 李俊霖

phase 1

- 容易看出在调用字符串比较函数之前载入了一个参数值，如果不匹配则爆炸。可知这是一个字符串匹配的关卡，若不输入正确的字符串就会爆炸。

```
(gdb) disassemble phase_1
Dump of assembler code for function phase_1:
   0x0000555555556439 <+0>:     sub    $0x8,%rsp
   0x000055555555643d <+4>:     lea    0x1d04(%rip),%rsi        # 0x555555558148
   0x0000555555556444 <+11>:    callq  0x555555556982 <strings_not_equal>
   0x0000555555556449 <+16>:    test   %eax,%eax
   0x000055555555644b <+18>:    jne    0x555555556452 <phase_1+25>
   0x000055555555644d <+20>:    add    $0x8,%rsp
   0x0000555555556451 <+24>:    retq
   0x0000555555556452 <+25>:    callq  0x555555556c5e <explode_bomb>
   0x0000555555556457 <+30>:    jmp    0x55555555644d <phase_1+20>
End of assembler dump.
(gdb) x/s 0x555555558148
0x555555558148: "NVIDIA has been the single worst company we've ever dealt with. So NVIDIA, ____ ___!"
```

- 输出载入地址处的字符串，
  为 `NVIDIA has been the single worst company we've ever dealt with. So NVIDIA, ____ ___!`，即为通关密码。
- ps.这是一句 `Linus Torvalds` 的名言。

phase 2

- 通过函数名和格式控制符都可以发现，要求输入六个数字。

```
(gdb) x/s 0x555555558615
0x555555558615: "%d %d %d %d %d %d"
```

- 要求第一个数为4，第二个数为0x11，即17。

```
  2472:   e8 a7 08 00 00          callq  2d1e <read_six_numbers>
  2477:   83 3c 24 04             cmpl   $0x4,(%rsp)
  247b:   75 07                   jne    2484 <phase_2+0x2b>
  247d:   83 7c 24 04 11          cmpl   $0x11,0x4(%rsp)
  2482:   74 05                   je     2489 <phase_2+0x30>
  2484:   e8 d5 07 00 00          callq  2c5e <explode_bomb>
  2489:   48 89 e3                mov    %rsp,%rbx
```

- 关键代码分析，经过寄存器之间的来回赋值，此处要满足数列的通项关系为$4a_n + a_{n+1} = a_{n+2}$。

```
249a:   74 14                   je     24b0 <phase_2+0x57>
249c:   8b 13                   mov    (%rbx),%edx
249e:   8b 43 04                mov    0x4(%rbx),%eax
24a1:   8d 04 90                lea    (%rax,%rdx,4),%eax
24a4:   39 43 08                cmp    %eax,0x8(%rbx)
24a7:   74 ea                   je     2493 <phase_2+0x3a>
```

- 根据前两项计算可得到该数列为 `4 17 33 101 233 637` ，为通关密码。

phase 3

- 通过查看scanf前载入的格式控制符可以发现，要求输入两个数字。

```
0x24cc <phase_3>        sub    $0x18,%rsp
0x24d0 <phase_3+4>      mov    %fs:0x28,%rax
0x24d9 <phase_3+13>     mov    %rax,0x8(%rsp)
0x24de <phase_3+18>     xor    %eax,%eax
0x24e0 <phase_3+20>     lea    0x4(%rsp),%rcx
0x24e5 <phase_3+25>     mov    %rsp,%rdx
0x24e8 <phase_3+28>     lea    0x2132(%rip),%rsi        # 0x4621
0x24ef <phase_3+35>     callq  0x2150 <__isoc99_sscanf@plt>
0x24f4 <phase_3+40>     cmp    $0x1,%eax
```

```
(gdb) x/s 0x4621
0x4621: "%d %d"
```

- 第一个数字不能大于7。

```
0x24f9 <phase_3+45>     cmpl   $0x7,(%rsp)
0x24fd <phase_3+49>     ja     0x25c3 <phase_3+247>
```

- 阅读代码逻辑发现第二个数字的比较是根据第一个数字来的，而且内部数字、逻辑比较复杂，直接输入两个数字 `2 456` 进入gdb调试。

```
   0x55555555655f <phase_3+147>    sub    %edx,%eax
   0x555555556561 <phase_3+149>    jmp    0x55555555652a <phase_3+94>
>  0x555555556563 <phase_3+151>    mov    0x4bb7(%rip),%edx        # 0x55555555b120
   0x555555556569 <phase_3+157>    mov    $0x2d7,%eax
   0x55555555656e <phase_3+162>    sub    %edx,%eax
   0x555555556570 <phase_3+164>    jmp    0x55555555652a <phase_3+94>
   0x555555556572 <phase_3+166>    mov    0x4ba8(%rip),%edx        # 0x55555555b120
   0x555555556578 <phase_3+172>    mov    $0x12f,%eax
   0x55555555657d <phase_3+177>    sub    %edx,%eax
   0x55555555657f <phase_3+179>    jmp    0x55555555652a <phase_3+94>
   0x555555556581 <phase_3+181>    mov    0x4b99(%rip),%edx        # 0x55555555b120
   0x555555556587 <phase_3+187>    mov    $0x17a,%eax
   0x55555555658c <phase_3+192>    sub    %edx,%eax
    ,--.!,
  __/   -*-              phase_3                          L??   PC: 0x555555556563
,d08b.  '|`
0088MM
`9MMP'
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Breakpoint 1, 0x0000555555556514 in phase_3 () ever dealt with. So NVIDIA, ____ ___!
(gdb) nidefused. (Interesting key, huh?)
0x0000555555556563 in phase_3 ()
(gdb) number 2.
2 459
```

- 经过操作，可以得知最后eax的值要为356，即第一个数为2时，输入的第二个数应为356才正确。

```
   0x55555555652e <phase_3+98>     test   %edx,%edx
   0x555555556530 <phase_3+100>    js     0x555555556536 <phase_3+106>
>  0x555555556532 <phase_3+102>    cmp    %eax,%edx
   0x555555556534 <phase_3+104>    je     0x55555555653b <phase_3+111>
   0x555555556536 <phase_3+106>    callq  0x555555556c5e <explode_bomb>
   0x55555555653b <phase_3+111>    mov    0x8(%rsp),%rax
   0x555555556540 <phase_3+116>    sub    %fs:0x28,%rax
   0x555555556549 <phase_3+125>    jne    0x5555555565d2 <phase_3+262>
   0x55555555654f <phase_3+131>    add    $0x18,%rsp
   0x555555556553 <phase_3+135>    retq
    ,--.!,
  __/   -*-              phase_3                          L??   PC: 0
(gdb) p $edx
$1 = 459
(gdb) ni
0x0000555555556530 in phase_3 ()
(gdb) ni
0x0000555555556532 in phase_3 ()
(gdb) p $edx
$2 = 459
(gdb) p $eax
$3 = 356
(gdb)
```

- 因此一个可行的通关密码为 `2 356` 。

phase 4

- 需要输入两个数字。

```
0x2630 <phase_4+28>     lea    0x1fea(%rip),%rsi        # 0x4621
0x2637 <phase_4+35>     callq  0x2150 <__isoc99_sscanf@plt>
0x263c <phase_4+40>     cmp    $0x2,%eax
0x263f <phase_4+43>     jne    0x2647 <phase_4+51>
0x2641 <phase_4+45>     cmpl   $0xe,(%rsp)
0x2645 <phase_4+49>     jbe    0x264c <phase_4+56>
0x2647 <phase_4+51>     callq  0x2c5e <explode_bomb>

exec No process In:
(gdb) x/s 0x4621
0x4621: "%d %d"
```

- 进入func4函数之后判断两个值，可知第二个数操作之后必须为6，且函数返回值必须为6。第一个数输入必须小于15。

```
e8 79 ff ff ff          callq  25d7 <func4>            #进入func4函数
8b 4c 24 04             mov    0x4(%rsp),%ecx
8d 51 fa                lea    -0x6(%rcx),%edx
89 54 24 04             mov    %edx,0x4(%rsp)
83 fa 06                cmp    $0x6,%edx              #第二个数操作之后必须为6
75 05                   jne    2673 <phase_4+0x5f>
83 f8 06                cmp    $0x6,%eax              #函数返回值必须为6
```

- 经过gdb调试得知，第二个数为12时，经过操作之后必定为6，与第一个数无关。
- 下一步分析func4函数，找到第一个数的输入。
- func4代码逻辑如图，可知这是一个递归函数。

```
0000000000025d7 <func4>:
    25d7:   48 83 ec 08             sub    $0x8,%rsp
    25db:   89 d0                   mov    %edx,%eax              #eax = edx
    25dd:   29 f0                   sub    %esi,%eax              #eax -=esi
    25df:   89 c1                   mov    %eax,%ecx              #ecx = eax
    25e1:   c1 e9 1f                shr    $0x1f,%ecx             #ecx = 0
    25e4:   01 c1                   add    %eax,%ecx              #ecx += eax
    25e6:   d1 f9                   sar    %ecx                   #ecx = ecx / 2
    25e8:   01 f1                   add    %esi,%ecx              #ecx += esi
    25ea:   39 f9                   cmp    %edi,%ecx
    25ec:   7f 0c                   jg     25fa <func4+0x23>      #ecx > edi -> edx = ecx - 1
    25ee:   b8 00 00 00 00          mov    $0x0,%eax
    25f3:   7c 11                   jl     2606 <func4+0x2f>      #ecx < edi -> esi = ecx + 1
    25f5:   48 83 c4 08             add    $0x8,%rsp
    25f9:   c3                      retq
    25fa:   8d 51 ff                lea    -0x1(%rcx),%edx
    25fd:   e8 d5 ff ff ff          callq  25d7 <func4>
    2602:   01 c0                   add    %eax,%eax
    2604:   eb ef                   jmp    25f5 <func4+0x1e>
    2606:   8d 71 01                lea    0x1(%rcx),%esi
    2609:   e8 c9 ff ff ff          callq  25d7 <func4>
    260e:   8d 44 00 01             lea    0x1(%rax,%rax,1),%eax
    2612:   eb e1                   jmp    25f5 <func4+0x1e>
```

- 使用gdb调试，发现输入第一个数6时，得到函数的返回值为6.
- 因此通关密码为 6 12 满足条件。

phase 5

- 进入函数，输入字符串，且字符个数一定要为6个。

```
0x2696 <phase_5+4>      callq  0x2965 <string_length>
0x269b <phase_5+9>      cmp    $0x6,%eax
0x269e <phase_5+12>     jne    0x26cc <phase_5+58>
```

- 可以发现不爆炸的条件为，循环6次，使最终的ecx为46。

```
0x5555555566b3 <phase_5+33>     movzbl (%rax),%edx
0x5555555566b6 <phase_5+36>     and    $0xf,%edx
0x5555555566b9 <phase_5+39>     add    (%rsi,%rdx,4),%ecx
0x5555555566bc <phase_5+42>     add    $0x1,%rax
0x5555555566c0 <phase_5+46>     cmp    %rdi,%rax
0x5555555566c3 <phase_5+49>     jne    0x5555555566b3 <phase_5+33>
```

- 根据代码可知，ecx 是一个累计的量，每次累计的值为[4(c&0xf)+rsi]（[]的含义为取该地址内的值）。
- 地址内的值可以打出来，如下：

```
(gdb) b *phase_5+12
phase_5
0x555555558200 <array.0>:       0x00000002
(gdb) x 93824992248324
0x555555558204 <array.0+4>:     0x0000000a
(gdb) x 93824992248328
0x555555558208 <array.0+8>:     0x00000006
(gdb) x 93824992248332
0x55555555820c <array.0+12>:    0x00000001
(gdb) x 93824992248336
0x555555558210 <array.0+16>:    0x0000000c
(gdb) x 93824992248340
0x555555558214 <array.0+20>:    0x00000010
(gdb) x 93824992248344
0x555555558218 <array.0+24>:    0x00000009
(gdb) x 93824992248348
0x55555555821c <array.0+28>:    0x00000003
(gdb) x 93824992248352
0x555555558220 <array.0+32>:    0x00000004
(gdb) x 93824992248356
0x555555558224 <array.0+36>:    0x00000007
```

```
(gdb) x 93824992248360
0x555555558228 <array.0+40>:    0x0000000e
(gdb) x 93824992248364
0x55555555822c <array.0+44>:    0x00000005
(gdb) x 93824992248368
0x555555558230 <array.0+48>:    0x0000000b
(gdb) x 93824992248372
0x555555558234 <array.0+52>:    0x00000008
(gdb) x 93824992248376
0x555555558238 <array.0+56>:    0x0000000f
(gdb) x 93824992248380
0x55555555823c <array.0+60>:    0x0000000d
```

- a的ASCII值为0x61，与0xf取与操作得1；b得2，c得3，以此类推。因此相当于每一个小写字母在表中从+4一直往下取对应的数值，6个对应的数值相加为46即可。
- 可满足的情况有多种，此处取一组 `abcdec` ，对应的数字分别为10 6 1 12 16 1，加起来等于46，满足条件。因此通关密码可以为 `abcdec` 。

phase 6

- 进入汇编代码phase_6函数，需要输入六个数字。
- 第一个数字-1必须<=5，即第一个必须<=6。

```
0x5555555567e7 <phase_6+269>    sub    $0x1,%eax
0x5555555567ea <phase_6+272>    cmp    $0x5,%eax
0x5555555567ed <phase_6+275>    ja     0x55555555671b <phase_6+65>
```

- 对一大段循环跳转进行分析，总结出判断合法性基本逻辑为：输入的6个数必须每个都不相同，且都在1-6区间内，即一个1-6数字的排列。

```
0x55555555672c <phase_6+82>     jg     0x5555555567d9 <phase_6+255>
0x555555556732 <phase_6+88>     mov    0x0(%r13,%rbx,4),%eax
0x555555556737 <phase_6+93>     cmp    %eax,0x0(%rbp)
```

- 接下来一段循环，含义为将x转成7-x。

```
2743:   48 8b 54 24 08          mov    0x8(%rsp),%rdx   #rsp+0x8的值赋给rdx
2748:   48 83 c2 18             add    $0x18,%rdx
274c:   b9 07 00 00 00          mov    $0x7,%ecx        # ecx = 7
2751:   89 c8                   mov    %ecx,%eax        # eax = 7
2753:   41 2b 04 24             sub    (%r12),%eax      # eax -= (r12)
2757:   41 89 04 24             mov    %eax,(%r12)      # (r12) = eax
275b:   49 83 c4 04             add    $0x4,%r12        # r12 += 4
275f:   4c 39 e2                cmp    %r12,%rdx        # if rdx != r12, loop
2762:   75 ed                   jne    2751 <phase_6+0x77>
```

- 在2772中，代码把一个地址载入了rdx中，将这个地方的地址打出来，可以发现有以下结构。

```
2764:   be 00 00 00 00          mov    $0x0,%esi
2769:   8b 4c b4 10             mov    0x10(%rsp,%rsi,4),%ecx
276d:   b8 01 00 00 00          mov    $0x1,%eax
2772:   48 8d 15 e7 eb 00 00    lea    0xebe7(%rip),%rdx        # 11360 <node1>
2779:   83 f9 01                cmp    $0x1,%ecx
277c:   7e 0b                   jle    2789 <phase_6+0xaf>
```

发现其类似于一个链表的结构，第一项为数据，第二项为下标顺序，第三项为next的地址，可以如node1的next为70512=0x11370，得到验证。由于只显示了node1-node5，按照这个思路把node5的next值打出来，可得到六个node，node6的next为0，即该链表有六个元素，也符合输入六个数的规则。

```
(gdb) x/24wd 0x11360
0x11360 <node1>:        117     1       70512   0
0x11370 <node2>:        175     2       70528   0
0x11380 <node3>:        61      3       70544   0
0x11390 <node4>:        665     4       70560   0
0x113a0 <node5>:        309     5       35488   0
0x113b0:        0       0       0       0
(gdb) x/3wd 35488
0x8aa0 <node6>: 703     6       0
```

- 下面对关键的linknode操作的代码进行分析。

```
#link_node key code
    2805:   48 8b 5b 08             mov    0x8(%rbx),%rbx
    2809:   83 ed 01                sub    $0x1,%ebp
    280c:   74 11                   je     281f <phase_6+0x145>
    280e:   48 8b 43 08             mov    0x8(%rbx),%rax
    2812:   8b 00                   mov    (%rax),%eax
    2814:   39 03                   cmp    %eax,(%rbx)
    2816:   7d ed                   jge    2805 <phase_6+0x12b>
    2818:   e8 41 04 00 00          callq  2c5e <explode_bomb>
```

发现此处是按照输入顺序进行考察。要求(%rbx) − %eax ≥ 0 推出 (%rbx) ≥ %eax即0x8(%rbx)，可知要按照node的数值进行降序排列，得到排列如下： `6 4 5 2 1 3` ，因此7-x之前的值为 `1 3 2 5 6 4` ，即为通关密码。

- 至此，前六关全部完成破解。

secret phase

- 首先寻找secret phase的入口，在汇编代码中发现secret phase函数，搜索发现在phase_defuse里会调用secret phase函数。

```
  2f4c:   48 8d 3d 2d 14 00 00    lea    0x142d(%rip),%rdi        # 4380 <array.0+0x180>
  2f53:   e8 18 f1 ff ff          callq  2070 <puts@plt>
  2f58:   b8 00 00 00 00          mov    $0x0,%eax
  2f5d:   e8 45 f9 ff ff          callq  28a7 <secret_phase>
  2f62:   eb ad                   jmp    2f11 <phase_defused+0x78>
  2f64:   e8 37 f1 ff ff          callq  20a0 <__stack_chk_fail@plt>
```

- 在调用之前，发现phase_4的输入格式控制符除了两个数字之外，多了一个字符串%s，这个就是隐藏入口；根据字符串比对函数，把比对的值打出，可知该隐藏的命令字符串是Testify。

```
  2ef4:   48 8d 35 70 17 00 00    lea    0x1770(%rip),%rsi        # 466b <array.0+0x46b>
  2efb:   48 8d 3d e6 ea 00 00    lea    0xeae6(%rip),%rdi        # 119e8 <input_strings+0x168>
  2f02:   b8 00 00 00 00          mov    $0x0,%eax
  2f07:   e8 44 f2 ff ff          callq  2150 <__isoc99_sscanf@plt>
  2f0c:   83 f8 03                cmp    $0x3,%eax
  2f0f:   74 1a                   je     2f2b <phase_defused+0x92>
  2f11:   48 8d 3d a0 14 00 00    lea    0x14a0(%rip),%rdi        # 43b8 <array.0+0x1b8>
  2f18:   e8 53 f1 ff ff          callq  2070 <puts@plt>
  2f1d:   48 8d 3d c4 14 00 00    lea    0x14c4(%rip),%rdi        # 43e8 <array.0+0x1e8>
  2f24:   e8 47 f1 ff ff          callq  2070 <puts@plt>
  2f29:   eb 9b                   jmp    2ec6 <phase_defused+0x2d>
  2f2b:   48 8d 7c 24 10          lea    0x10(%rsp),%rdi
  2f30:   48 8d 35 3d 17 00 00    lea    0x173d(%rip),%rsi        # 4674 <array.0+0x474>
  2f37:   e8 46 fa ff ff          callq  2982 <strings_not_equal>
  2f3c:   85 c0                   test   %eax,%eax
```

```
(gdb) x/s 0x4674
0x4674: "Testify"
(gdb) x/s 0x43e8
0x43e8: "Your instructor has been notified and will verify your solution."
(gdb) x/s 0x466b
0x466b: "%d %d %s"
```

- 对secret_phase函数分析。

```
00000000000028a7 <secret_phase>:
    28a7:   48 83 ec 18             sub    $0x18,%rsp
    28ab:   c7 44 24 0c 1e 00 00    movl   $0x1e,0xc(%rsp)
    28b2:   00
    28b3:   e8 a7 04 00 00          callq  2d5f <read_line>
    28b8:   48 89 c6                mov    %rax,%rsi
    28bb:   48 8d 3d 1e 62 00 00    lea    0x621e(%rip),%rdi        # 8ae0 <t0>
    28c2:   e8 7c ff ff ff          callq  2843 <fun7>              # 进入fun7函数
    28c7:   8b 54 24 0c             mov    0xc(%rsp),%edx
    28cb:   39 c2                   cmp    %eax,%edx                #if eax == 30，安全，结束
    28cd:   75 16                   jne    28e5 <secret_phase+0x3e>
```

  - 进入fun7函数前把一段地址传给了rdi。
  - 从fun7函数返回后，返回值若等于0x1e，则安全，否则爆炸。

- 对fun7函数分析。

```
0002843 <fun7>:
   55                   push   %rbp
   53                   push   %rbx
   48 83 ec 08          sub    $0x8,%rsp
   48 89 fb             mov    %rdi,%rbx         #某段地址的头指针 rdi rbx
   48 89 f5             mov    %rsi,%rbp         #更新的next字符的指针 -> rbp
   48 85 ff             test   %rdi,%rdi
   74 2b                je     287f <fun7+0x3c>   #为零就炸?
   0f b6 55 00          movzbl 0x0(%rbp),%edx     #c的ascii
   84 d2                test   %dl,%dl           #c的ascii
   74 2a                je     2886 <fun7+0x43>   #为零就  jmp3
   80 fa 61             cmp    $0x61,%dl          #char == 'a'?
   74 29                je     288a <fun7+0x47>   #char == 'a',  jump1

   0f be d2             movsbl %dl,%edx           #char != 'a'
   83 ea 61             sub    $0x61,%edx         #char -= 'a'
   b8 01 00 00 00       mov    $0x1,%eax          # eax = 1
   39 d0                cmp    %edx,%eax
   74 1f                je     288f <fun7+0x4c>   # if edx == eax (char == 'b'),jmp4 继续递归读取下一个字
   83 c0 01             add    $0x1,%eax          # char != 'b',eax++
   83 f8 1a             cmp    $0x1a,%eax         # 不能超过z 否则爆炸
   75 f4                jne    286c <fun7+0x29>
   e8 e1 03 00 00       callq  2c5e <explode_bomb>
   eb 21                jmp    28a0 <fun7+0x5d>    #到这里 进不来?  jump2
   e8 da 03 00 00       callq  2c5e <explode_bomb>
   eb ce                jmp    2854 <fun7+0x11>
   8b 03                mov    (%rbx),%eax         #jmp3 to,  此时的rbx地址里的值->返回值,要为30
   eb 16                jmp    28a0 <fun7+0x5d>    #到这里才不会继续递归 jump2
   ba 00 00 00 00       mov    $0x0,%edx              #jump1 to
   48 8d 75 01          lea    0x1(%rbp),%rsi         #第二个字母 到rsi  jmp4 to
   48 63 d2             movslq %edx,%rdx              # 0
   48 8b 7c d3 08       mov    0x8(%rbx,%rdx,8),%rdi  # 8*rdx(和a的差值)+rbx+8 放到 rdi，对更新
   e8 a3 ff ff ff       callq  2843 <fun7>            # 递归
   48 83 c4 08          add    $0x8,%rsp              #递归返回到这里就退出;  jump2 to
   5b                   pop    %rbx
   5d                   pop    %rbp
   c3                   retq
```

fun7函数的基本逻辑是：

- 循环读入每一个字符，每一个只能是小写字母，否则爆炸。
- 循环找到每一个字符和字母‘a’的差值x，x*8作为字节偏移量，把这个偏移量加上进入fun7函数前的地址（基地址），该地址处存储的值作为下一个字符的基地址，循环往复，相当于一个用偏移和内存里的值建立一连串链式链接的结构（整体上是一个类似于**字典树**的结构）。
- 对于最后一个字母，它偏移之后的地址里存放的值应该为函数返回值，即0x1e。

- 开始解构该链式结构。
  - 运行后使用gdb调试找到该段内存的初始地址 `0x55555555cae0 <t0>:`
  - 使用类似于 `(gdb) x/3000gx 0x55555555cae0` 的指令将该位置前后的大量内存以8字节的形式打出来，如图。

```
0x55555555c1d0 <t144+208>:      0x0000000000000000      0x0000000000000000
0x55555555c1e0 <t175>:  0x000000af0000001c      0x0000000000000000
0x55555555c1f0 <t175+16>:       0x0000000000000000      0x0000000000000000
0x55555555c200 <t175+32>:       0x0000000000000000      0x0000000000000000
0x55555555c210 <t175+48>:       0x0000000000000000      0x0000000000000000
0x55555555c220 <t175+64>:       0x0000000000000000      0x0000000000000000
0x55555555c230 <t175+80>:       0x0000000000000000      0x0000000000000000
0x55555555c240 <t175+96>:       0x0000000000000000      0x0000000000000000
0x55555555c250 <t175+112>:      0x0000000000000000      0x0000000000000000
0x55555555c260 <t175+128>:      0x0000000000000000      0x0000000000000000
0x55555555c270 <t175+144>:      0x0000000000000000      0x0000000000000000
0x55555555c280 <t175+160>:      0x0000000000000000      0x0000000000000000
0x55555555c290 <t175+176>:      0x0000000000000000      0x0000000000000000
0x55555555c2a0 <t175+192>:      0x0000000000000000      0x0000000000000000
0x55555555c2b0 <t175+208>:      0x0000000000000000      0x0000000000000000
0x55555555c2c0 <t61>:   0x0000003d00000008      0x0000000000000000
0x55555555c2d0 <t61+16>:        0x0000000000000000      0x0000000000000000
```

- 寻找目标值0000001e，然后一步步从后往前逆推，直到回到初始地址 `<t0>` 。得到以下回溯链条：

```
0x55555555cae0 <t0>:
0x55555555cb80 <t0+160>:        0x000055555555cbc0      0x0000000000000000    20

0x55555555cbc0 <t69>:   0x0000004500000000      0x000055555555d480            1

0x55555555d480 <t95>:
0x55555555d510 <t95+144>:       0x000055555555d560      0x0000000000000000    18

0x55555555d560 <t180>:
0x55555555d570 <t180+16>:       0x000055555555d640      0x0000000000000000    2

0x55555555d640 <t181>:  0x000000b500000000      0x000055555555d720            1

0x55555555d720 <t182>:
0x55555555d780 <t182+96>:       0x000055555555d800      0x0000000000000000    12

0x55555555d800 <t183>:
0x55555555d860 <t183+96>:       0x000055555555b300      0x0000000000000000    12

0x55555555b300 <t184>:  0x000000b80000001e      0x0000000000000000
```

- 每一个节点的相对于‘a’的偏移量分别为 `20 1 18 2 1 12 12` ，对应的字母分别为 `t a r b a l l` ，即该字符串为 `tarball` ，为该关卡的密码。

- **`tarball` 是linux系统下一种方便的打包工具。**
- 所有炸弹已拆除，结果如图。

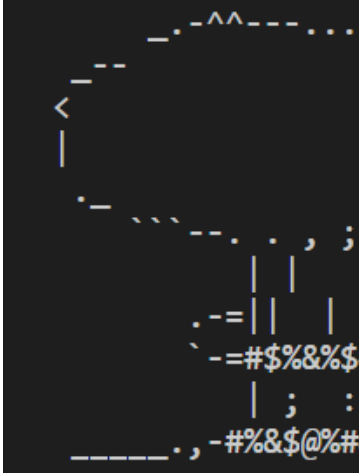
```
[2021201709@work122 bomblab]$ ./bomb
     ,--.!,
   __/   -*-
 ,d08b.  '|`
 0088MM
 `9MMP'
 Welcome to my fiendish little bomb. You have 6 phases with
 which to blow yourself up. Have a nice day!
 NVIDIA has been the single worst company we've ever dealt with. So NVIDIA, ____ ___!
 Phase 1 defused. (Interesting key, huh?)
 4 17 33 101 233 637
 That's number 2.
 2 356
 Halfway there!
 6 12 Testify
 So you got that one.  Try this one.
 abcdec
 Good work!  On to the next...
 1 3 2 5 6 4
 Curses, you've found the secret phase!
 But finding it and solving it are quite different...
 tarball
 Wow! You've defused the secret stage!
 Congratulations! You've defused the bomb!
 Your instructor has been notified and will verify your solution.
```

总结与体会

- 经过本次$bomblab$的实践，我对汇编代码的阅读从一开始的带有一丝恐惧和抗拒心理到后来的比较熟练，可以熟练地应用gdb调试方法对自己阅读代码得到的逻辑假设和猜想做求证。大胆猜测，小心求证，是完成本次实验的八字箴言。
- 对6个普通 `phase` 和 `secret phase` 的求解过程，我对链表、树等数据结构和循环、递归函数的汇编实现方式有了更为深入的理解。
- 稍显遗憾的是，由于在gdb调试过程中刚开始的不熟练外加某两次单步调试ni误输入为n，导致拆弹过程中炸弹很不幸地炸了3次。
- 炸得确实很绚烂！

```
Single stepping until exit from function phase_6,
which has no line number information.

By my efflux of deep crimson, topple this white world! EXPLOSION!!!
```



```
The bomb has blown up.
Your instructor has been notified.
[Inferior 1 (process 5892) exited with code 010]
(gdb)
```