Phase_1:

第一题我们通过分析可以发现，炸弹触发的条件是输入了与预存的字符串不同的字符串，所以我们要解开炸弹只需要找到 strings_not_equal 这个函数调用的用于对比的字符串在什么地方即可。首先我们通过观察可以进行一下猜测，第二行的代码为 lea 0x17c1(%rip),%rsi ，这一行代码将%rip 偏移 0x17c1(%rip)后的地址存储在%rsi 中，那么我们猜测这个%rsi 中的地址中存储的就是所用来对比的字符串，下面来进行一次尝试

```
Reading symbols from bomb...done.
(gdb) b phase_1
Breakpoint 1 at 0x1204
(gdb) b explode_bomb
Breakpoint 2 at 0x1996
(gdb) run
Starting program: /home/jovyan/CSAPP/10225501403/bomb54/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
test

Breakpoint 1, 0x0000555555555204 in phase_1 ()
(gdb) next
Single stepping until exit from function phase_1,
which has no line number information.

Breakpoint 2, 0x0000555555555996 in explode_bomb ()
(gdb) info registers rsi
rsi            0x5555555569d0   93824992242128
(gdb) x/s 0x5555555569d0
0x5555555569d0: "I am for medical liability at the federal level."
(gdb)
```

很显然，我们猜出了答案的位置，接下俩我们进入 strings_not_equal 中去验证一下。

```
00000000000016db <strings_not_equal>:
    16db:   41 54                   push   %r12
    16dd:   55                      push   %rbp
    16de:   53                      push   %rbx
    16df:   48 89 fb                mov    %rdi,%rbx
    16e2:   48 89 f5                mov    %rsi,%rbp
```

经过查看，发现该函数的两个输入参数一个位于%rdi 寄存器一个位于%rsi 寄存器，符合我们的猜想，第一题结束。

Phase_2

第二题的汇编代码开始部分进行了一系列准备，在 0x123d 处调用了 read_six_numbers 这一函数，通过观察后面的代码我们可以发现该函数的返回值保存在%rsp 寄存器中，换句话说，我们输入的 6 个数字保存在爱%rsp 中，接下来进行了一个对比，将%rsp 中的第一个数和立即数 0 进行了对比，若不等于 0，则会跳转到 0x124f，即炸弹爆炸，那么这要求我们输入的第一个数字必须是 0。接下来我们发现代码又比较了%rsp 中的第二个数和 1，若相同则跳转到 0x1254，不同则继续运行下一行，即炸弹爆炸。这要求我们输入的第二个数字必须是 1。我们看到 0x1245，这一行将%rsp 保存的值保存到%rbx，下一行计算栈上地址 0x10(%rbx) 并保存到 %rbp 寄存器，然后无条件跳转到 1266。1266 处是关键的操作，这一步将%rbx 寄存器加上 4 偏移处的值移动到 %eax 寄存器，然后将将 %eax 寄存器与 %rbx 寄存器中的值相加，结果保存在 %eax 寄存器，下一步比较 %eax 寄存器的值与 0x8(%rbx) 处的值。看到这里我们就可以大致确定这个数列的规律了，即下一项等于前两项之和，那么可以得到我们的输入结果：0 1 1 2 3 5，下面进行验证。

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am for medical liability at the federal level.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
```

验证通过，成功拆除第二个炸弹。

Phase_3

第三题考差了分支。前八行汇编代码是为后续操作压栈和为 sscanf 函数传参，我们通过查看 rsi 寄存器中地址发现其为"%d %d"，这表明我们要输入两个整数，接下来我们又发现我们输入的第一个数字不能大于 7，因为如果大于 7，会直接跳转到炸弹。后面我们发现，该程序通过计算，将我们输入的第一个参数转化为一个地址，这个地址就是通过我们的输入计算出的跳转到的情况，分析后发现共有七种情况，分别对应第一个输入为 1,2,3,4,5,6,7，然后根据分支给 eax 赋值为不同的数值，最后比价 eax 和输入的第二个数，那么到这里我

```
mov     $0x1cd,%eax
jmp     1322 <phase_3+0x8f>
mov     $0x182,%eax
jmp     1322 <phase_3+0x8f>
mov     $0x26e,%eax
jmp     1322 <phase_3+0x8f>
mov     $0x36b,%eax
jmp     1322 <phase_3+0x8f>
mov     $0x56,%eax
jmp     1322 <phase_3+0x8f>
mov     $0x2c6,%eax
jmp     1322 <phase_3+0x8f>
mov     $0x37c,%eax
jmp     1322 <phase_3+0x8f>
```

们就发现一共有七种答案，我们输入其中一种，1 461，验证发现炸弹被成功拆除。

```
That's number 2.  Keep going!
1 461

Breakpoint 2, 0x0000555555555293 in phase_3 ()
(gdb) continue
Continuing.
Halfway there!
```

Phase_4

第四题开头和第三题一样，也是初始化输入，调用的输入规则也是和第三题一样，即输入两个整数，然后检测输入的数字个数是否为 2，输入的第一个数字和 15 进行比较，如果小于等于就跳转开炸弹进行下一步。接下来我们发现该程序在一系列准备后调用了 func4，并且将 func4 返回的结果和 31 进行比较，如不同则引爆，然后检测第二个输入的数字是否为 31，不是则引爆，那么关键就在于这个 func4 的作用，我们查看 func4 的汇编代码。查看后我们发现，该函数是一个递归调用自身的函数，函数中有两个重要的数值，一个存储在 edx 另一个存储在 esi，而在递归调用前，会进行一个运算，即计算 eax=edx/2+esi，然后根据这个数值去进行递归，直到 eax 等于我们输入的第一个数字，此时我们注意到每一步的 eax 存储在 ebx 中，最后当结束递归时，返回的数值为所有 ebx 的和。了解逻辑后我们可以解得当我们的输入为 13 的时候恰好 func4 的输出为 31，这样我们就得到了第五题的答案 13 31，验证后发现通过。

```
Halfway there!
13 31

Breakpoint 3, 0x000055555555537b in phase_4 ()
(gdb) continue
Continuing.
So you got that one.  Try this one.
```

Phase_5

第五题开头如前，初始化输入两个整数，检测输入结果，如果数量小于等于 1 就引爆炸弹，接下来会进行一个循环，通过分析发现，rsi 存储了一个数组的起始地址，我们暂且用 rsi[k] 来表示这个数组中第 k+1 个数字，那么这个循环就是不断令 eax=rsi[eax]，直到 eax 等于 15 时退出循环，同时 ecx 为每一步 eax 的和，退出循环后，检测循环次数是否为 15，ecx 是否和我们输入的第二个数字相等，那么我们的关键就在于找到这个数组的数值，根据题意来看这是一个包含了 16 个整型的数组，我们通过 gdb 进行调试，查看该数组。

```
(gdb) x/128bx 0x5555555556a60
0x5555555556a60 <array.3417>:    0x0a    0x00    0x00    0x00
    0x02    0x00    0x00    0x00
0x5555555556a68 <array.3417+8>:  0x0e    0x00    0x00    0x00
    0x07    0x00    0x00    0x00
0x5555555556a70 <array.3417+16>: 0x08    0x00    0x00    0x00
    0x0c    0x00    0x00    0x00
0x5555555556a78 <array.3417+24>: 0x0f    0x00    0x00    0x00
    0x0b    0x00    0x00    0x00
0x5555555556a80 <array.3417+32>: 0x00    0x00    0x00    0x00
    0x04    0x00    0x00    0x00
0x5555555556a88 <array.3417+40>: 0x01    0x00    0x00    0x00
    0x0d    0x00    0x00    0x00
0x5555555556a90 <array.3417+48>: 0x03    0x00    0x00    0x00
    0x09    0x00    0x00    0x00
0x5555555556a98 <array.3417+56>: 0x06    0x00    0x00    0x00
    0x05    0x00    0x00    0x00
```

这样我们就获得了数组，由于最后一个数字为 15，且要 15 步，那么我们根据数组反推即可，最终得到输入为 5 时第 15 步为 15，计算每一步的和为 115，那么我们输入 5 115 验

```
So you got that one.  Try this one.
5 115
Good work!  On to the next...
```

证发现通过。

Pahse_6

这个题目首先进行了初始化，然后我们输入六个数字的地址存储到 rsp，然后对输入的六个数字进行检测，即检测有无重复，是否都大于等于 1 小于等于 6，检测后我发现

```
150d:   8b 0c b4              mov    (%rsp,%rsi,4),%ecx
1510:   b8 01 00 00 00        mov    $0x1,%eax
1515:   48 8d 15 14 2d 20 00  lea    0x202d14(%rip),%rdx
151c:   83 f9 01              cmp    $0x1,%ecx
151f:   7f d2                 jg     14f3 <phase_6+0x70>
```

第 1515 行处，该汇编代码计算了 rip+0x202d14，并存储至 rdx，我们打印出 rdx 观察

```
(gdb) x/3wx 0x555555758230
0x555555758230 <node1>: 0x00000362      0x00000001      0x55758240
(gdb) x/3wx 0x55758240
0x55758240:     Cannot access memory at address 0x55758240
(gdb) x/3wx 0x555555758240
0x555555758240 <node2>: 0x00000360      0x00000002      0x55758250
(gdb) x/3wx 0x555555758250
0x555555758250 <node3>: 0x0000023e      0x00000003      0x55758260
(gdb) x/3wx 0x555555758260
0x555555758260 <node4>: 0x000001b4      0x00000004      0x55758270
(gdb) x/3wx 0x555555758270
0x555555758270 <node5>: 0x000000c3      0x00000005      0x55758110
(gdb) x/3wx 0x555555758280
0x555555758280 <host_table>:    0x55556d0f      0x000055550x00000000
(gdb) x/3wx 0x555555758220
0x555555758220 <n34+16>:        0x557580f0      0x000055550x00000000
(gdb) x/3wx 0x555555758290
0x555555758290 <host_table+16>: 0x00000000      0x000000000x00000000
(gdb) x/3wx 0x555555758110
0x555555758110 <node6>: 0x00000247      0x00000006      0x00000000
(gdb) q
```

很明显，rdx 处存储的是一个节点，一个节点包括三个部分，数值、id、下一个节点的地址，我们都打印后发现一共是 6 个节点，同时发现后面的汇编代码是将节点的数值根据我们的输入的数字的顺序排序后进行检测，

```
156b:   48 8b 5b 08           mov    0x8(%rbx),%rbx
156f:   83 ed 01              sub    $0x1,%ebp
1572:   74 11                 je     1585 <phase_6+0x102

1574:   48 8b 43 08           mov    0x8(%rbx),%rax
1578:   8b 00                 mov    (%rax),%eax
157a:   39 03                 cmp    %eax,(%rbx)
157c:   7e ed                 jle    156b <phase_6+0xe8>

157e:   e8 13 04 00 00        callq  1996 <explode_bomb>
```

检测是否为从小到大排序，那么显然，我们只需要输入这六个节点从小到大的排序序号即可，即 5 4 3 6 2 1，验证发现，正确。

```
Good work!  On to the next...
5 4 3 6 2 1
Congratulations! You've defused the bomb!
```

Secret phase

完成上述 phase 后发现汇编代码后面还有一个 secret phase，通过查找发现，是在 phase_defused 中调用，查看 phase_defused，

```
1bdd:   48 8d 7c 24 10        lea    0x10(%rsp),%rdi
1be2:   48 8d 35 1f 11 00 00  lea    0x111f(%rip),%rsi
```

我们查看在该步的 rdi 和 rsi 中的地址，发现 rdi 中是第四题的输入，rsi 为 DrEvil，这样我发现在第四题答案后输入 DrEvil 后就可以 call secret phase，接下来看到 secret phase，我们发现在准备后调用了 fun7，并且 fun7 要返回 4，我们查看 fun7，发现 fun7 是对于一个树做操作，数的头节点地址存于 rdi 中，输入的数大于则返回值*2+1，移动到右子，反之则*2，移动到左子，那么 4=2*（2*（2*0+1）），即从根节点开始，向左向左向右即可得到我们的答案，我们操作如下

```
(gdb) x/5wx 0x555555758150
0x555555758150 <n1>:    0x00000024      0x00000000      0x55758170 0x00005555
0x555555758160 <n1+16>: 0x55758190
(gdb) x/5wx 0x555555758170
0x555555758170 <n21>:   0x00000008      0x00000000      0x557581f0 0x00005555
0x555555758180 <n21+16>:        0x557581b0
(gdb) x/5wx 0x5555557581f0
0x5555557581f0 <n31>:   0x00000006      0x00000000      0x55758030 0x00005555
0x555555758200 <n31+16>:        0x55758090
(gdb) x/5wx 0x555555758030
0x555555758030 <n41>:   0x00000001      0x00000000      0x00000000 0x00000000
0x555555758040 <n41+16>:        0x00000000
(gdb) x/5wx 0x555555758090
0x555555758090 <n42>:   0x00000007      0x00000000      0x00000000 0x00000000
0x5555557580a0 <n42+16>:        0x00000000
(gdb) continue
Continuing.
```

这样，我们得到了答案，0x7，也就是 7，验证答案

```
Starting program: /home/jovyan/CSAPP/10225501403/bomb54/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I am for medical liability at the federal level.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
1 461
Halfway there!
13 31 DrEvil
So you got that one.  Try this one.
5 115
Good work!  On to the next...
5 4 3 6 2 1
Curses, you've found the secret phase!
But finding it and solving it are quite different...
7
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
[Inferior 1 (process 100) exited normally]
(gdb)
```

至此所有炸弹均被拆除。