bomblab 报告

姓名：吴双琦

学号：2024201575

总分	phase_1	phase_2	phase_3	phase_4	phase_5	phase_6	secret_phase
7	1	1	1	1	1	1	1

scoreboard 截图：

2024201575	0	0	0	0	0	0	0

解题报告

phase_1

```
"Someday, I'll conquer the land and have you slain."
```

讲解题目思路

第一题注意到调用了<strings_not_equal>，并且前一句将某个值赋值给了%rsi。 又知函数参数的顺序一般是：第一个参数放在%rdi，第二个参数放在%rsi，因此可以推断出%rsi里现在存的正是目标字符串的首地址。用x/s $rsi读取即可得到答案。

phase_2

```
366001 878939 413345 1005476
```

讲解题目思路

已知int sscanf(const char *str, const char *format, ...)从字符串读取格式化输入，返回成功匹配和存储的个数。所以我们查询存储格式字符串的%rsi，发现需要我们输入的格式为"%d %d %d %d"，为四个整数，分别存储在%rdx,%rcx,%r8,%r9指向的地址中（即%rsp,%rsp+4,%rsp+8,%rsp+12）。 接下来的代码执行了二重循环。%rdi存储的是某一个int类型的数组A，该数组有3列，在外部循环每次对%rdi中的地址+12可以实现对每行元素遍历，而内部循环每次+4可以实现对同一行的列的遍历。%rbx中存储的也是某一个int类型的数组B的首地址，在内部循环的时候每次对%rbx的地址+8可以得到同一行连续的列的值，而外部循环对%rbx+4使得整体切换到第2行，重复内部循环，得到第二行的每列值。%ecx中存储每一次内部循环对应相乘累加的和，如此重复两次，所以phase_2完成了矩阵A和矩阵B的相乘操作。 从1502开始，代码实现了对比输入的四个答案和标答的功能。%rsp~%rsp+12存储了输入，而%rsp+16~%rsp+28存储了（代码中用%rbp~%rbp+16表示）标准答案（A的某一行点乘B的某一列的结果）。我们只需要获取%rsp+16~%rsp+28的值即可。

phase_3

```
7 89
```

讲解题目思路

首先获取输入格式（方法同上）,发现需要输入两个整数. sscanf将把第一个输入放到%rdx存的地址中,由上文代码知也就是存在%rsp存的地址中;第二个输入放在%rcx存的地址中,%rcx存的地址是%rsp+4. 接着比较%rsp中的值是否等于7,所以第一个输入值为7. 往下找,代码让%edx存储%rsp+4这个地址中存的值,最后会比较%eax和%edx中的值是否相等.合理推断第二个输入的答案存在%eax中.倒推回去找%eax的值(直接中gdb调试到这一步)得到%eax中存的是89.所以第二个输入为89.

phase_4

```
31 "AB"
```

讲解题目思路

首先获取输入格式字符串"%d %2s",得到答案需要一个整数和一个最大宽度为2的字符串. 由代码知输入1\2分别存在%rsp存的地址+12/16中. 接下来将%rdi赋值为5,进入func4_1.func4_1是一个递归函数,返回一个整数,递推式为ri = 2r_(i-1) + 1, r1 = 1, r0 = 0. 所以当i=5时,最终返回31.返回后代码将31与%rsp+12中的值(即输入1)比较,说明第一个输入的整数为31. 接下来把存放输入2的地址%rsp+16放进%rdi,然后调用string_length验证输入2长度是否为2. 往下找最终比较字符串相等的代码,gdb查看$rsi存的字符串,得到第2个输入应该为字符串"AB" --以上为直接得到答案的过程,下面接着分析汇编-- fun4_2接受6个参数,分别存在%rdi,%rsi,%rdx,%rcx,%r8,%r9中.phase_4传入了(5,0x1a,0x41,0x43,0x42,%rsp+20的地址) fun4_2在arg_1==1时,将%rdx和%rcx的低8位分别写入arg6即某一char指针的前1,2个字节并返回; 在arg_1!=1时,首先调用func4_1(new_arg1 = arg_1 - 1),比较返回值res和arg_2的大小 如果返回值res小于arg_2,则比较返回值res+1与arg_2是否相等,如果是则将%rdx和%rcx的低8位分别写入arg6即某一char指针的前1,2个字节并返回;否则交换arg_3和arg_5的位置,递归调用func4_2(new_arg_1, arg_2-res-1,arg_5,arg_4,arg_3,arg_6) 否则(res>=arg_2)交换arg_4和arg_5的位置,递归调用func4_2(new_arg_1, arg_2, arg_3,arg_5,arg_4,arg_6) 通过计算可以得到最终返回时%rbp的前两个字节分别存储的0x41,0x42 抽象来看,arg_1限制了最大递归深度,arg_2则是是否递归调用交换参数位置的决定参数,arg_3,4,5存了3个char类型常量0x41,0x42,0x43,最终有两个将存入arg_6指针指向的char类型数组中

phase_5

```
">74598"
```

讲解题目思路

首先由string_length函数调用后的比较可知答案是长度为6的字符串 从最后的字符串比较倒推可知,目标字符串("bruins")放在%rsi中,我们的解答放在$rdi中.倒推发现解答的六个字符分别放在%rsp+1/2/3/4/5/6中.再向上看,%rsp+i的值通过循环设置.%rdx为循环计数.输入的字符串存在%rbx中.假设%rdx对应变量d,d = 0/1/2/3/4/5,解答数组a[6],起始位置为%rsp+1.输入字符串为b[6] 每次循环,a[i] = *((b[i]+15)%16 + %rcx地址),此处%rcx应该为某个字符串数组,通过公式可以找到对应位置的字符串.反解得到输入字符串. 通过gdb找到%rcx中存的字符

串"maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"."bruins"对应位置在13,6,3,4,8,7,对应14,7,4,5,9,8,同时加16的任意倍数(取48),即得输入字符串的ascii码,翻译过来是">74598"

phase_6

```
2 3 5 6 1 4
```

讲解题目思路

首先从read_six_numbers里读取格式字符串，发现本题接受六个整数"%d %d %d %d %d %d"。由该函数可知，phase_6向read_six_numbers传入的第二个参数为存储输入字符串的指针，在phase_6中该指针存在%r13中. 接下来是一个两层循环，外层先检查输入的六个整数值是否（无符号jmpa表示无符号大于时爆炸）小于等于6，内层循环则遍历输入数组检查是否和外层值相等，相当于检查唯一性。 检查完输入后，代码实现基于输入顺序找到对应链表节点，并将对应节点按输入顺序排序的功能（第i个输入为j，则新排序中第j个节点为第i个原节点）然后通过遍历节点比较前后节点大小，确保新顺序中各节点以非降序排列。检查原链表（起始节点为node1:9220）发现节点1-6中存储的值依次为648->134->268->750->398->587.则确保非降序的输入应该为2 3 5 6 1 4.

secret_phase

```
33113
```

讲解题目思路

这里找secret_phase主要是查看phase_defused函数，发现当num_input_strings==6时，从保存输入的地址加载一个字节到 %cl，并零扩展到 %ecx，然后检查%cl是否为0检查这个字节是否为空，当非空时循环检查输入的每个字节，cmp $0x20,%cl 检查该字节是否时空格，非空时检查空格计数%edx是否大于5。如果空格数>=6，则跳转到218e检查,当空格数为6时，比较了输入和一个字符串。这说明开启secret_phase的关键时在phase_6答案后面加空格和一个验证字符串。gdb调试得到验证字符串为"hidden"，顺利开启secret_phase. 进入后： 首先确定输入字符串长度为20以内.然后调用func_7,func_7接受4个参数，第一个参数为字符串，后3个参数为int类型的整数，分别表示当前x,y位置和递归深度。

```
000000000000197d <func7>:
    197d:   48 81 ec 98 00 00 00    sub    $0x98,%rsp //预留栈空间
    1984:   89 f0                   mov    %esi,%eax //copy arg_2 to %eax
    1986:   41 89 c9                mov    %ecx,%r9d //copy arg_4 to %r9d,以便下一步使用%rcx
    1989:   64 48 8b 0c 25 28 00    mov    %fs:0x28,%rcx
    1990:   00 00
    1992:   48 89 8c 24 88 00 00    mov    %rcx,0x88(%rsp)//将段寄存器存的数赋值到0x88(%rsp)
    1999:   00
    199a:   31 c9                   xor    %ecx,%ecx //清零
   //硬编码方向数组dir[4][8]
    1a9a:   00
    1a9b:   83 fe 04                cmp    $0x4,%esi
```

```
    1a9e:   75 6b                   jne    1b0b <func7+0x18e>
    1aa0:   83 fa 07                cmp    $0x7,%edx
    1aa3:   75 66                   jne    1b0b <func7+0x18e>//if(arg2 != 4 ||
arg_3 != 7)移动
    1aa5:   49 63 c9                movslq %r9d,%rcx //if(arg2 == 4 && arg_3 == 7)
成功到达目标点位
    1aa8:   0f b6 34 0f             movzbl (%rdi,%rcx,1),%esi //取str[arg_4]
    1aac:   b9 01 00 00 00          mov    $0x1,%ecx
    1ab1:   40 84 f6                test   %sil,%sil
    1ab4:   74 34                   je     1aea <func7+0x16d>
    1ab6:   b9 00 00 00 00          mov    $0x0,%ecx
    1abb:   41 83 f9 13             cmp    $0x13,%r9d
    1abf:   7f 29                   jg     1aea <func7+0x16d>//if(str[arg_4]==0 ||
arg_4 > 19)return
    1ac1:   41 89 f2                mov    %esi,%r10d //if(str[arg_4]!=0 && arg_4
<= 19)
    1ac4:   41 83 e2 07             and    $0x7,%r10d //str[arg_4]%8
    1ac8:   83 e6 07                and    $0x7,%esi //str[arg_4]%8
    1acb:   41 89 c0                mov    %eax,%r8d //x位置
    1ace:   44 03 04 b4             add    (%rsp,%rsi,4),%r8d //当前x位置+硬编码方
向数组第一行第str[arg_4]列
    1ad2:   41 89 d3                mov    %edx,%r11d
    1ad5:   44 03 5c b4 20          add    0x20(%rsp,%rsi,4),%r11d//当前y位置+硬编
码方向数组第二行第str[arg_4]列
    1ada:   44 89 c6                mov    %r8d,%esi
    1add:   44 09 de                or     %r11d,%esi
    1ae0:   b9 00 00 00 00          mov    $0x0,%ecx
    1ae5:   83 fe 07                cmp    $0x7,%esi
    1ae8:   76 3f                   jbe    1b29 <func7+0x1ac> //if(str[arg_4]%8 <=
7)检验该步正确性（走日字）
    1aea:   48 8b 84 24 88 00 00    mov    0x88(%rsp),%rax //检查位置是否越界
    1af1:   00
    1af2:   64 48 2b 04 25 28 00    sub    %fs:0x28,%rax
    1af9:   00 00
    1afb:   0f 85 9e 00 00 00       jne    1b9f <func7+0x222>
    1b01:   89 c8                   mov    %ecx,%eax
    1b03:   48 81 c4 98 00 00 00    add    $0x98,%rsp
    1b0a:   c3                      ret //return
    1b0b:   b9 00 00 00 00          mov    $0x0,%ecx //未到达(4,7)
    1b10:   41 83 f9 13             cmp    $0x13,%r9d
    1b14:   7f d4                   jg     1aea <func7+0x16d>
    1b16:   49 63 c9                movslq %r9d,%rcx
    1b19:   0f b6 34 0f             movzbl (%rdi,%rcx,1),%esi
    1b1d:   b9 00 00 00 00          mov    $0x0,%ecx
    1b22:   40 84 f6                test   %sil,%sil
    1b25:   74 c3                   je     1aea <func7+0x16d> //if(arg_4 > 19 ||
str[arg_4] == 0)return;
    1b27:   eb 98                   jmp    1ac1 <func7+0x144>//合法进入下一阶段移动
    1b29:   4d 63 d2                movslq %r10d,%r10
    1b2c:   42 03 44 94 40          add    0x40(%rsp,%r10,4),%eax //原x位置+dx日字
偏移
    1b31:   42 03 54 94 60          add    0x60(%rsp,%r10,4),%edx //原y位置+dy日字
偏移
    1b36:   48 8d 35 73 36 00 00    lea    0x3673(%rip),%rsi        # 51b0 <row0>
```

```
//加载棋盘
    1b3d:   85 c0                   test   %eax,%eax
    1b3f:   7e 0b                   jle    1b4c <func7+0x1cf> //偏移后x<=0
    1b41:   48 8b 76 08             mov    0x8(%rsi),%rsi //x>0
    1b45:   83 c1 01                add    $0x1,%ecx //arg_4+1
    1b48:   39 c8                   cmp    %ecx,%eax
    1b4a:   75 f5                   jne    1b41 <func7+0x1c4>
    1b4c:   48 63 d2                movslq %edx,%rdx
    1b4f:   b9 00 00 00 00          mov    $0x0,%ecx
    1b54:   80 3c 16 01             cmpb   $0x1,(%rsi,%rdx,1) //检验棋盘By上障碍是
否蹩马脚
    1b58:   74 90                   je     1aea <func7+0x16d>//蹩马脚则return
    1b5a:   48 8d 15 4f 36 00 00    lea    0x364f(%rip),%rdx        # 51b0 <row0>
    1b61:   45 85 c0                test   %r8d,%r8d
    1b64:   7e 11                   jle    1b77 <func7+0x1fa>
    1b66:   b8 00 00 00 00          mov    $0x0,%eaxv
    1b6b:   48 8b 52 08             mov    0x8(%rdx),%rdx
    1b6f:   83 c0 01                add    $0x1,%eax
    1b72:   41 39 c0                cmp    %eax,%r8d
    1b75:   75 f4                   jne    1b6b <func7+0x1ee>
    1b77:   49 63 c3                movslq %r11d,%rax
    1b7a:   b9 00 00 00 00          mov    $0x0,%ecx
    1b7f:   80 3c 02 01             cmpb   $0x1,(%rdx,%rax,1)//检验棋盘Bx上障碍是否
蹩马脚
    1b83:   0f 84 61 ff ff ff       je     1aea <func7+0x16d>//蹩马脚则return
    1b89:   41 8d 49 01             lea    0x1(%r9),%ecx
    1b8d:   44 89 da                mov    %r11d,%edx
    1b90:   44 89 c6                mov    %r8d,%esi
    1b93:   e8 e5 fd ff ff          call   197d <func7>//递归 (str, new_x, new_y,
depth+1)
    1b98:   89 c1                   mov    %eax,%ecx
    1b9a:   e9 4b ff ff ff          jmp    1aea <func7+0x16d>
    1b9f:   e8 fc f4 ff ff          call   10a0 <__stack_chk_fail@plt>
    ```
```

检查代码可以发现func_7实现的是中国象棋中马的走法，即走日字形，且使用硬编码的4*8数组分别记录x,y方向及复核是否出现马蹩脚情况的对应移动方式。而障碍物地图则以链表的形式给出，gdb调试查看链表内容，可以找到形如下表的障碍物地图，当下一步的x,y加上复核数组中对应值，会经过蹩脚障碍点时，该步不能顺利完成.需要输入的字符串给出每一步移动方向的编号，使得最终能到达(4,7)。画图后不难找出一条路径，对应答案33113，能够顺利到达(4,7)

```
00010000
01000000
00010001
00000001
00000100
01000001
00000000
00000100
```

......

反馈/收获/感悟/总结

<!-- 这一节，你可以简单描述你在这个 lab 上花费的时间/你认为的难度/你认为不合理的地方/你
```

```
认为有趣的地方 -->

<!-- 或者是收获/感悟/总结 -->

<!-- 200 字以内，可以不写 -->

## 参考的重要资料

<!-- 有哪些文章/论文/PPT/课本对你的实现有重要启发或者帮助，或者是你直接引用了某个方法 -->

<!-- 请附上文章标题和可访问的网页路径 -->
```