BOMBLAB实验报告

姓名：蔡亦扬
学号：23307130258

成功截图

```
stevencai@StephenCai:~/lab2/lab2-bomblab-Caibao7$ cat password.txt - | ./bomb++
Please enter your Student ID (23307xxxxxx) in the config.txt file.
Note: Different Student IDs will generate different answers. Therefore, do not attempt to use someone else's ID for the
answers.
You have 6 phases with which to blow yourself up. Have a nice day!
PHASE 1...
Phase 1 defused. How about the next one?PHASE 2...
That's number 2. Keep going!PHASE 3...
Halfway there!PHASE 4...
So you got that one. Try this one.PHASE 5...
Good work! On to the next...PHASE 6...
Cool! your skill on Reverse Engineer is great.Congratulations!
Welcome to the secret phase of Bomb++!
You are really a Master of Reverse Engineer!You have successfully defused the bomb!


---------- Your score: 85 ----------

The remaining 15 points are determined by your report and the format of password.txt.
```

phase_1

phase_1的汇编代码首先根据我的ID_hash进行一系列运算，得到了一个基于ID_hash的偏移量phase_1_offset，将 phase_1_offset 加到基地址 phase_1_str 上，得到目标字符串的内存地址。查看地址可以看到目标字符串nes. AI's unchecked growth risks losing human control。

phase_2

phase_2一开始先分配栈空间，然后调用了构造函数 _ZZ7phase_2ENUt_C2Ev 来初始化缓冲区。然后调用 read_six_numbers 函数，将用户输入的六个数字存储到缓冲区中。之后进入循环，每次循环计算一个临时值用来判断是否与输入值相等。
公式为：temp = a × buffer[i−1] + b。a由缓冲区第七个整数设置（-10），b是缓冲区第八个整数，即 (ID_hash & 3) + 1。
因此构造输入为1 -6 64 -636 6364 -63636满足递推要求，可以通过。

phase_3

开头先分配栈帧，处理ID_hash（操作好后存入-0x8(%rbp)，我的是0x3），读取输入并检测（先确保输入3个值）。
接着根据输入的第一个值（-0x10(%rbp)，存入eax）来进行初步的跳转：除去会爆炸的跳转，其它的跳转情况不符合我的0x3，因此考虑接下来的代码块（此时第一个输入应该在0到34之间）。
接下来的这个代码块会根据第一个输入值，进行一个地址的运算并完成跳转。跳转后的基本过程是比较第二个输入值与一个设定好的常数，如果相等比较ID与一个常数，如果相等则跳转到代码后面的部分，比较第三个输入与一个储存在寄存器里的值，如果相等则通过，而前三个比较如果有一个不相等则爆炸。
所以我们应该跳转到比较ID为0x3的代码块，然后去判断这个分支下对应的输入2和输入3是什么。为了跳转到这里，我们可以设置断点并一步步ni来检测。鉴于输入1的可能取值只是0到34，我们可以枚举输入1。当枚举到13的时候，成功跳转进了0x3的代码块，看到了输入2应该为21，再次跳转后通过查看-0x1(%rbp)的值知道了此时这里存的是0x72，也就是114，对应小写字母 'r'。故最后应该输入13 21 r

phase_4

首先读入一个数，处理成64位，分割高32位和低32位分别存在-0x4(%rbp)和-0x8(%rbp)，然后检测这两个数是否在[1, 10]这个区间，这个检测是通过两个寄存器来储存月结标志实现的。
之后进入函数_ZL3CIEi，这个函数是一个递归函数，可以写出这个函数的C代码：

```
int _ZL3CIEi(int arg) {
if (arg == 0) {
return 1;
} else {
int temp = _ZL3CIEi(arg / 2);
if (arg & 1) {
return (temp * temp) * 8;
} else {
return temp * temp;
}
}
}
```

函数return以后是比较return值与0x40000000，所以应该让return值与0x40000000相等。倒推可以得到输入的arg应该是10。
故整个的输入应该满足高32位为10，低32位处在[1, 10]的区间，故可以输入42949672961。

phase_5

首先输入三个值，根据第一个值决定进三个函数中的一个("behavior" 对应 AIBehaviorRegulator,"ethics" 对应 AIEthicsRegulator，"growth" 对应 AIGrowthRegulator)，然后设置值与74 (0x4A)比，如果小于就爆，从这里看出只能是growth。然后回跳转到一个虚函数，提前断点并disassemble发现这个函数最后会比较第二个输入与2034($0x7f2)，不相等会爆，相等继续到_ZN11AIRegulator18is_phase5_passableEj函数，这个函数会比较第三个输入与ID_hash的低12位，我的值是(315)。最后输入
growth 2034 315即可通过。

phase_6

先处理读入的六个数字，输入数字要在0-6的范围内。然后调用build_stack来创建一个栈，stackBottom存在0x5555559f6340。再调用maintain_monotonic_sequence来检查。
可以写出maintain_monotonic_sequence的C代码(伪代码)：

```
int maintain_monotonic_sequence(Node **stack_ptr, int number) {
int stackBottom = address; // 从0x5555559f6340加载 stackBottom 的值
if (number < stackBottom) {
return 0;
}
// 当堆栈指针未到达 stackBottom 时
while(xxx) {
int top = ttt; // 获取堆栈顶部的值
```

```
    if (top < number) {
        // 如果顶部值小于 number，弹出堆栈
        stack_pop(stack_ptr);
```

```
    } else {
        // 否则，找到合适的位置，停止弹出
        break;
    }
}
stack_push(stack_ptr, number);
return 1;
```

}
在build_stack后设置断点，检查到0x4a2340存放的值是4，那么根据代码的含义，输入从大于等于4的数开始的非递减序列就可以了，比如4 4 4 5 6 6。

secret_phase

进入隐藏关需要修改secret_key的值使之非0，这里通过输入一个非常大的字符串来overflow实现。隐藏关中的汇编代码中有一个循环，循环次数为 3 次，每次循环处理输入字符串的一个字符。对每个字符 c，计算 (c - 'A') & 1。如果结果不为0则爆。意味着字符 c 必须是 A, C, E, G, I, J 中的一个。然后进入状态跳转的模块，最后的状态必须要为3，由跳转规则：
初始状态 (state = 0)：
期望字符为 'C'，将 state 更新为 1。
如果字符为 'A'，将 state 更新为 4（爆）
状态 1：
期望字符为 'I'，将 state 更新为 2。
如果字符为 'C'，将 state 更新为 4（爆）。
状态 2：
期望字符为 'E'，将 state 更新为 3。
推出输入应该为CIE，成功实现跳转。