计算机系统基础

实验报告

Lab-2-BombLab

姓名：马颢宸

学号：23307110426

一、实验目的

综合运用 x86 汇编语言 / Linux / gdb / 数据结构等的相关知识
解决一系列挑战，提升逆向工程技能

二、实验结果

关闭剧情模式的拆弹截图

```
root@Ranxiaoxiao:~/icslab/lab2/lab2-bomblab-ranxiaoxiao-mmm# 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.
```

三、实验过程

phase_1

phase_1 主要考察函数调用相关知识。本题主要采用静态分析函数功能，动态分析匹配参数。

由 ID_hash 可知，此题答案存在学号偏置。

由函数 string_not_equal 可知，此题应输入字符串，并与目标字符串匹配。

分析（86-101）易知-0x18(%rbp)和-0x8(%rbp)应分别存储输入字符串与目标字符串。故直接查看$rdx 中的值，得目标字符串 ines. AI's unchecked growth risks losing human control

故最后答案为 **ines. AI's unchecked growth risks losing human control**

phase_2

phase_2 主要考察循环相关知识。本题主要采用静态分析函数功能，动态分析匹配参数。

由函数 read_six_numbers 可知，此处应传入 6 个数字。

由（55、132-140）可知，将-0x4(%rbp)作为计数器，循环操作 5 次。

分析（64-97），存在语句 imul 和 add，即将当前参数与一固定数相乘，再与另一固定数相加，故直接查看-0x28(%rbp)与-0x24(%rbp)的值，可知是将参数与-10 相乘后再与 3 相加。

分析（100-122），可知是将上述操作后的值与下一参数进行比较，若相等则循环下一参数；若不等，则 BOMB！

检验，尝试输入第一个参数为 1，动态查看后续每个参数的值，符合上述推测规律。

故最后答案为 **1 -7 73 -727 7273 -72727**

phase_3

phase_3 主要考察分支相关知识。本题主要采用静态分析函数功能，动态分析匹配参数。

由函数 scanf 可知，此处应传入 3 个参数。

先观察汇编代码发现，函数中存在结构相似的 8 段，且每段中都有语句 cmpl 将

-0x8(%rbp)与 0-7 分别比较，故可知-0x8(%rbp)中即存储 case 的分支条件。且由函数 ID_hash 可知，-0x8(%rbp)应为学号偏置，故直接查看其中值得 6。

对应 case（6）汇编代码，分析（81-144）可知，当且仅当第一个参数为 233 时，能进入 case（6）。分析（383）可知，第二个参数应为 377。结合（376、459-473）可知，当且仅当第三个参数为 111 时，拆弹成功。但动态分析后发现，整数 111 会导致 BOMB！故联想至 ASCII 码，111 对应 o，输入后拆弹成功。

故最后答案为 **233 377 o**

特别地，此处对 8 个 case 中第三个参数的值进行分析后发现，八个 case 中对应字符相连得 overflow。即为 secret_phase 的开启条件。

phase_4

phase_4 主要考察递归相关知识。本题主要采用静态分析函数功能。

由 scanf 函数可知，此题输入一个参数。

第二部分（62-123）由 sar $0x20,%rax 可知，参数至少大于 32 位，故猜测参数为 64 位。由（66-77）可知，将参数高 32 位与低 32 位分开存储、运算。分析（80-123）可知，分别判断高位和低位的取值范围，只有当高位和低位都大于 0，小于等于 10 时，才能进入下一部分。否则 BOMB！

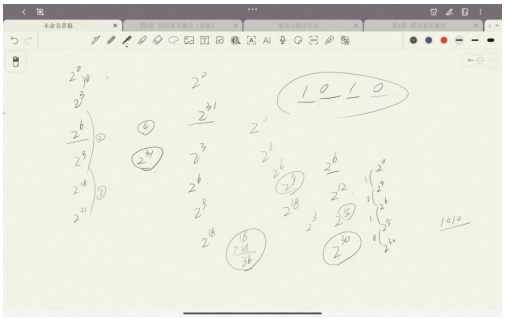
第三部分（_ZL3CIEi（15-35））将高位作为参数传入函数_ZL3CIEi。查看函数_ZL3CIEi 的汇编代码，可知_ZL3CIEi 即为递归函数，递归停止条件为传入参数为 0，递归操作为将参数算术右移 1 位。由第二部分取值范围易知，此处最多递归操作 4 次。

第三部分（_ZL3CIEi（40-71））分析可知，判断当前参数的最低位是否为 1。若为 1，则将%eax 平方；若为 0，则将%eax 平方后再左移三位。因为在最后一次递归时，将%eax 赋值为 1，故此处即做 1 的左移操作。

第四部分（142-150）分析易知，当函数_ZL3CIEi 返回值等于$0x40000000 时拆弹成功，结合上一部分可知，需将 1 左移 30 位。分析后，当且仅当高位为 1010 时，才能满足拆弹要求。

故最后答案为 **42949672970**

---> 00000000000000000000000000001010000000000000000000000000000001010



最后一步分析草稿

phase_5

phase_5 主要考察面向对象中虚函数的运用。由于上学期未选修面向对象课程，故特地先学习了虚函数的相关知识。本题主要采用静态分析函数功能，动态分析匹配参数。

分析 phase_5 主函数汇编代码后发现，可将代码分作四个部分。

第一部分（0-54）建栈 ---> 与拆弹答案无关，可忽略

第二部分（59-74）由先前的经验可知，调用 scanf 函数后返回的应是参数个数。由 cmp 语句和$0x3 可知，此处应输入三个值。

第三部分（79-238）分析可知，此处应是选择虚函数指针。即存在三个类的对象，根据输入选择调用哪个虚函数指针。此部分代码又可分为功能相同的三段，分别将第一个参数与三个值进行比较。由函数 strcmp 可看出，参与比较的参数应为字符串。分别查看 83、136、189 行的寄存器$rdx 可知，目标匹配的三个字符串分别是 behavior、ethics、growth。字符串匹配成功后会进入函数_Znwm 和函数_ZN17AIEthicsRegulatorC2Ev，主要进行虚函数指针选择功能，最后返回值$rbx 放入-0x18(%rbp)，即为指针。特别地，三个字符串按照顺序依次进行匹配；匹配成功则进入第四部分，失败则换到下一字符串；若三个字符串均匹配失败，则 BOMB！

第四部分（243-323）根据上一部分获得的虚函数指针，得到需要调用的虚函数*$rcx。（后续以输入字符串 growth 后的程序为示例）在虚函数中匹配第二个参数。此处通过 si 和 disas 进入虚函数并得到虚函数的汇编代码，由语句 cmpl $0x7f2,-0xc(%rbp) 可知，第二个参数应传入 2034。成功后，进入函数_ZN11AIRegulator18is_phase5_passableEj 匹配第三个参数。函数中 27-38 行应是进行学号偏置，故直接查看 38 行中寄存器$eax 的值可知，此处应传入 3798。

故最后答案为 **growth 2034 3798**

```
0x00005555555d5d05 <+243>:   mov    -0x18(%rbp),%rax  //-0x18（rbp） ---> 类的对象中的虚指针vptr
0x00005555555d5d09 <+247>:   mov    (%rax),%rax
0x00005555555d5d0c <+250>:   add    $0x10,%rax   //rax --->  虚表vtable（+10选择第十个虚函数）
0x00005555555d5d10 <+254>:   mov    (%rax),%rcx
0x00005555555d5d13 <+257>:   mov    -0x2c(%rbp),%edx
0x00005555555d5d16 <+260>:   mov    -0x18(%rbp),%rax
0x00005555555d5d1a <+264>:   mov    %edx,%esi
0x00005555555d5d1c <+266>:   mov    %rax,%rdi
0x00005555555d5d1f <+269>:   call   *%rcx    // --->  调用虚函数
0x00005555555d5d21 <+271>:   test   %eax,%eax
0x00005555555d5d23 <+273>:   je     0x5555555d5d3a <phase_5+296>
0x00005555555d5d25 <+275>:   mov    -0x30(%rbp),%edx
0x00005555555d5d28 <+278>:   mov    -0x18(%rbp),%rax
0x00005555555d5d2c <+282>:   mov    %edx,%esi
0x00005555555d5d2e <+284>:   mov    %rax,%rdi
0x00005555555d5d31 <+287>:   call   0x5555555d6162 <_ZN11AIRegulator18is_phase5_passableEj>
0x00005555555d5d36 <+292>:   test   %eax,%eax //eax! =0
0x00005555555d5d38 <+294>:   jne    0x5555555d5d41 <phase_5+303> //腹行
0x00005555555d5d3a <+296>:   mov    $0x1,%eax
0x00005555555d5d3f <+301>:   jmp    0x5555555d5d46 <phase_5+308>
0x00005555555d5d41 <+303>:   mov    $0x0,%eax  //希望
0x00005555555d5d46 <+308>:   test   %al,%al
0x00005555555d5d48 <+310>:   je     0x5555555d5d4f <phase_5+317>
0x00005555555d5d4a <+312>:   call   0x5555555d1e1f <explode_bomb>
0x00005555555d5d4f <+317>:   nop
0x00005555555d5d50 <+318>:   mov    -0x8(%rbp),%rbx
0x00005555555d5d54 <+322>:   leave
```

分析时，对第四部分的汇编代码注释

phase_6

phase_6 主要考察单调栈。本题主要采用静态分析函数功能。

由函数 read_six_numbers 可看出，此处应输入 6 个数字。

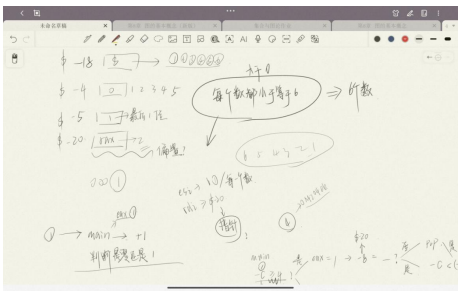
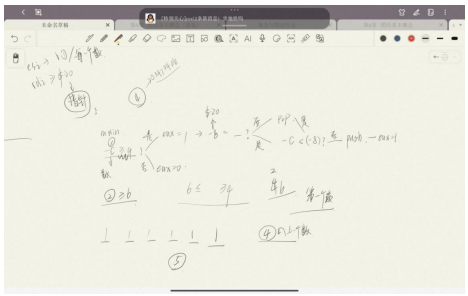
第一部分（49-124）主要判断 6 个输入数字的取值范围，以-0x4(%rbp)作为计数器。先判断输入数是否小于等于 6；成立，则判断输入数是否大于 0。成立，则进入下一部分。

第二部分（126-135）建栈 ---> 与拆弹答案无关，可忽略

第三部分（139-209）根据判断数字，为 0x5(%rbp) 赋值。为分析成立条件，先分析第四部分（211-229）可知，只有当 -0x5(%rbp) 时才能成功拆弹。又因为分析（184-205）可知 -0x5(%rbp) 的值取决于上一阶段 -0x5(%rbp) 的值，即只有当 -0x5(%rbp) 内存储值一直为 1 时，才能拆弹成功。

进入函数 maintain_monotonic_sequence 内部可知，栈内原存储值为 4，将栈内值与第一个数比较。当参数大于等于栈内值时，将参数 push 入栈内，并返回 1。故，可知第一个参数应大于等于 4，后面 5 个参数应大于等于上一个参数。

故最后答案为 **4 5 6 6 6 6**

根据汇编代码所绘流程图

特别地，为满足 secret_phase 的开启条件，此处在 password.txt 文件中，后续附上一连串的空格，以满足溢出。

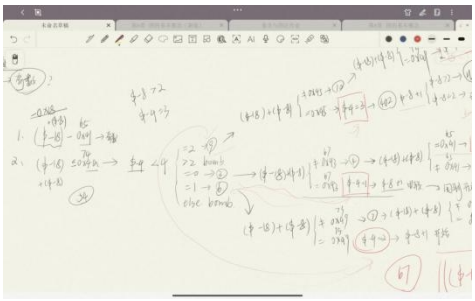
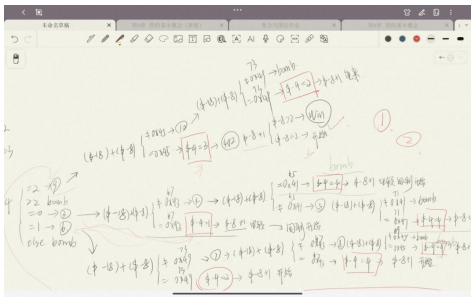
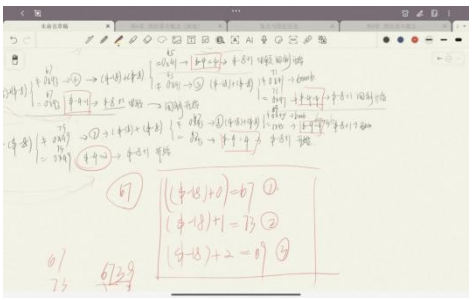
secret_phase

secret_phase 的开启条件在 phase_3 中隐含，即 phase3 所有 case 中对应的字母按顺序排列 ---> overflow。根据 overflow 的提示，再参考 main.cpp 函数中判断 sizeof（input），可合理推测开启条件为在某一题的答案中输入超过 88 位溢出。本题主要采用静态分析函数功能。

开启 secret_phase 后，观察汇编代码发现，整体逻辑比较简单清晰，主要是条件判断以及 jump 和条件赋值语句。结构相似功能相似的代码共有十段，交替分析输入的不同数值。

根据分析后发现，只有当输入连续为 67、73、69 时才能拆弹。根据 phase_3 的经验，联想 ASCII 码。

查表后得知，最后答案为 **CIE**

根据汇编代码所绘流程图

四、参考资料

https://blog.csdn.net/qq_42048450/article/details/117282640?fromshare=blogdetail&sharetype=blogdetail&sharerId=117282640&shareRefer=PC&sharesource=2402_84096879&sharefrom=from_link

五、对实验的感受

这是第二个 lab，总体感觉难度适中。最困扰的地方应该就是输入字符和字符串的 phase_3 和 phase_5 以及 secret_phase。这三个 phase 都是在分析出字符对应 ASCII 码后以为应输入数字，而困扰了 2 个多小时。最后还是灵感爆发才想起 ASCII 码。还有就是 phase_5 的虚函数，应该没有学过面向对象而感到在分析代码时比较吃力。其余 phase 都在静态动态分析结合下快速完成。

六、对助教的建议

如果后面还有类似的 lab，希望能设置参数类型的提醒 TAT，实在是被字符薄纱太多次了。希望后面的实验不要太难 www。设置 lab 请手下留情 QAQ。