

计算机系统基础实验 ICS 2020 秋季学期

Bomblab

1. 简介

本实验通过要求你使用课程所学知识拆除一个“binary bombs”来增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。一个“binary bombs”（**32位**二进制炸弹，下文将简称为炸弹）是一个 Linux 可执行程序，包含了 7 个阶段（或层次、关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信即解除了，否则炸弹“爆炸”打印输出“BOOM!!!”。实验的目标是拆除尽可能多的炸弹层次。

每个炸弹阶段考察了机器级程序语言的一个不同方面，难度逐级递增：

- 阶段 1：字符串比较
- 阶段 2：循环
- 阶段 3：条件/分支
- 阶段 4：递归调用和栈
- 阶段 5：指针
- 阶段 6：链表/指针/结构

另外还有一个隐藏阶段（树型结构），只有当你在第 4 阶段的解后附加一特定字符串后才会出现。

为完成二进制炸弹拆除任务，你需要使用 gdb 调试器和 objdump 来反汇编炸弹的可执行文件并跟踪调试每一阶段的机器代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。比如在每一阶段的开始代码前和引爆炸弹的函数前设置断点。

实验语言：C

实验环境：Linux

2. 实验步骤

2.1. 第一步：获取 bomb

在浏览器中打开 <http://47.92.205.8:15213>，在二进制炸弹请求表格中输入你的学号和邮箱地址（学号必须是真实学号，**邮箱地址合法即可，不必是真实的邮箱**），点击 Submit 按钮（注意不要反复点击）。服务器会构造属于你的炸弹，并以 tar 文件的形式 bombXXXXXXXXXX.tar 返回给你，其中 XXXXXXXXXXXX 是一个你的 bomb 的唯一标识，即

学号。注意：每个学号只能下载一次 bomb 文件，否则会显示 “You have already owned a bomb”。

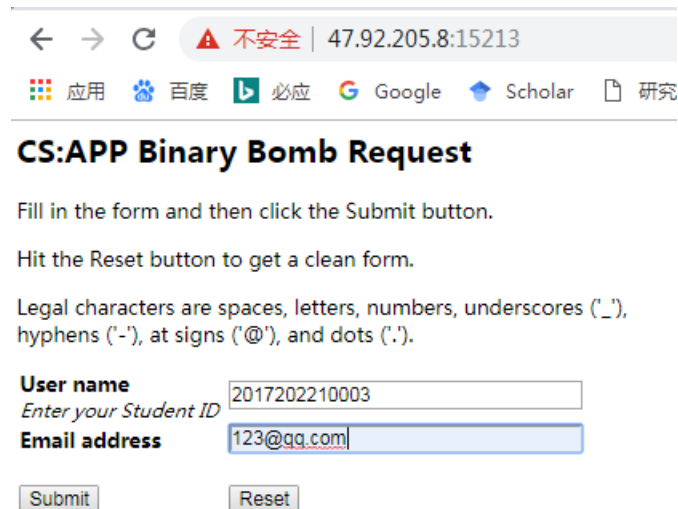


图 1. Bomb 文件请求页面

解压该 tar 文件（`tar -xvf bombXXXXXXXXXX.tar`）得到一个目录 `./bombXXXXXXXXXX`，其中包含如下文件：

- README：标识该 bomb 和所有者。
- bomb：bomb 的可执行程序。
- bomb.c：bomb 程序的 main 函数。

2.2. 第二步：拆除 bomb

本实验的任务就是拆除炸弹。一定要在指定的虚拟机上完成作业，在其他的环境上运行有可能导致失败。

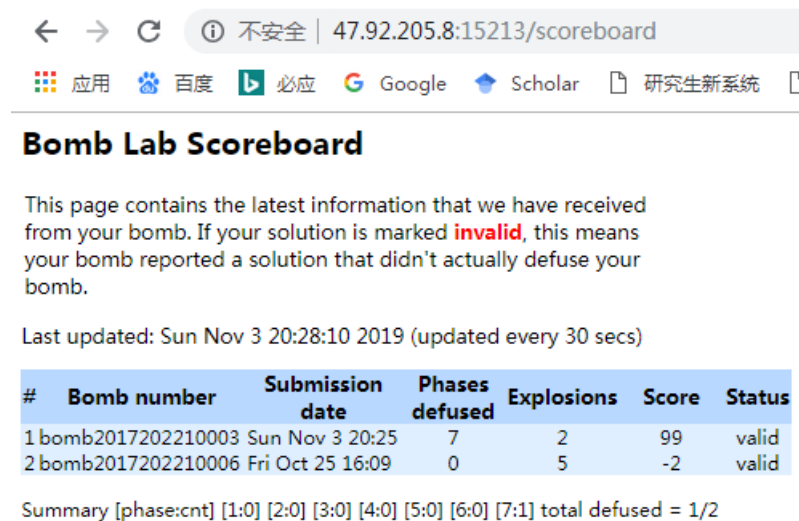
运行 `./bomb` 可执行程序需要 0 或 1 个命令行参数（详见 `bomb.c` 源文件中的 `main()` 函数）。如果运行时不指定参数，则该程序打印出欢迎信息后，期望你按行输入每一阶段用来拆除炸弹的字符串，根据你当前输入的字符串决定你是通过相应阶段还是炸弹爆炸导致任务失败。你也可将拆除每一阶段炸弹的字符串按行组织在一个文本文件中，然后作为运行程序时的唯一一个命令行参数传给程序，程序读入文件中的每一行直到遇到 EOF，再转到从 `stdin` 等待输入。这样对于你已经拆除的炸弹，就不用每次都重新输入，**只用放进文件里即可**。

前四个阶段每个 10 分，第五个阶段 20 分，第六个阶段 20 得分，第七个阶段（隐藏关）20 分。每输入错误一次，炸弹爆炸，会扣除 0.5 分，最多扣除 20 分，即所有阶段的爆炸次数之和超过 40 次之后，爆炸不再继续被扣分。**你必须小心！要学会单步跟踪调试汇编代码以及学会设置断点**。你还要学会如何检查寄存器和内存状态。很好的使用调试器是你在未来的职业生涯中赚到更多 money 的一项重要技能。

2.3. 实验结果提交

这是一项独立实验，每个人单独完成。**不需要单独提交最后的结果，运行或调试 bomb 程序会自动发送结果到服务器**（提交结果时，请确保你调试执行的虚拟机能够连接上网！），

可以在 <http://47.92.205.8:15213/scoreboard> 查看所有人的成绩结果。强烈建议：在**没有了解 bomb 运行机制之前断网**，以免尝试性的输出引爆炸弹，并送至服务器端被扣分！！



#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb2017202210003	Sun Nov 3 20:25	7	2	99	valid
2	bomb2017202210006	Fri Oct 25 16:09	0	5	-2	valid

Summary [phase:cnt] [1:0] [2:0] [3:0] [4:0] [5:0] [6:0] [7:1] total defused = 1/2

图 2. Bomb Lab Scoreboard

3. 提示！

下面简要说明完成本实验所需要的一些实验工具：

gdb

为了从二进制可执行程序“./bomb”中找出触发 bomb 爆炸的条件，可使用 GDB 来帮助对程序的分析。gdb 是 GNU 开源组织发布的一个强大的交互式程序调试工具。一般来说，gdb 主要帮你完成下面几方面的功能（更详细描述可参看 GDB 文档和相关资料）：

- 装载、启动被调试的程序。
- 让被调试的程序在你指定的调试断点处中断执行，方便查看程序变量、寄存器、栈内容等运行现场数据。
- 动态改变程序的执行环境，如修改变量的值。

objdump -d

该命令可用来对 bomb 中的二进制代码进行反汇编。通过阅读汇编源代码可以发现 bomb 是如何运行的。但是，objdump -d 不能告诉你 bomb 的所有信息，例如一个调用 sscanf 函数的语句可能显示为：8048c36: e8 99 fc ff ff call 80488d4 <_init+0x1a0>，你还需要 gdb 来帮助你确定这个语句的具体功能。

objdump -t

该命令可以打印出 bomb 的符号表。符号表包含了 bomb 中所有函数、全局变量的名称和存储地址。你可以通过查看函数名得到一些目标程序的信息。

strings

该命令可以显示二进制程序中的所有可打印字符串，例如：

```
strings bomb
```

实验步骤提示（以 bomb2017202210003 为例）

首先，每个同学拿到自己的 bomb 之后，可以先查看 bomb.c 文件。

```
65
66     /* Do all sorts of secret stuff that makes the bomb harder to defuse. */
67     initialize_bomb();
68
69     printf("Welcome to my fiendish little bomb. You have 6 phases with\n");
70     printf("which to blow yourself up. Have a nice day!\n");
71
72     /* Hmm... Six phases must be more secure than one phase! */
73     input = read_line();          /* Get input */
74     phase_1(input);              /* Run the phase */
75     phase_defused();             /* Drat! They figured it out! */
76     /* Let me know how they did it. */
77     printf("Phase 1 defused. How about the next one?\n");
78
79     /* The second phase is harder. No one will ever figure out
80      * how to defuse this... */
81     input = read_line();
82     phase_2(input);
83     phase_defused();
84     printf("That's number 2. Keep going!\n");
85
86     /* I guess this is too easy so far. Some more complex code will
87      * confuse people. */
88     input = read_line();
89     phase_3(input);
90     phase_defused();
91     printf("Halfway there!\n");
92
93     /* Oh yeah? Well, how good is your math? Try on this saucy problem! */
94     input = read_line();
95     phase_4(input);
96     phase_defused();
97     printf("So you got that one. Try this one.\n");
98
99     /* Round and 'round in memory we go, where we stop, the bomb blows! */
100    input = read_line();
101    phase_5(input);
102    phase_defused();
103    printf("Good work! On to the next...\n");
104
105    /* This phase will never be used, since no one will get past the
106     * earlier ones. But just in case, make this one extra hard. */
107    input = read_line();
108    phase_6(input);
109    phase_defused();
110
111    /* Wow, they got it! But isn't something... missing? Perhaps
112     * something they overlooked? Mua ha ha ha ha! */
113
114    return 0;
115 }
```

图 3. bomb.c 中 main 函数

这个告诉大家，炸弹运行的机制。read_line 获得输入 input，再将 input 传递给 phase_1 函数，然后执行 phase_1 函数，如果输入正确，phase_1 拆弹成功，接下来进入 phase_2，以此类推。

下面以第一关为例介绍实验步骤，注意演示的步骤只是提供一种思路，每个同学的 bomb 文件随机生成，拿到的 bomb 文件不同，反汇编后的结果也不同。

首先输入“objdump -d bomb > bombasm.txt”对 bomb 进行反汇编并将汇编源代码输出到“bombasm.txt”文本文件中。查看 bombasm.txt 文件，找到 main 函数，可以发

现 main 中调用了 phase_1, phase_1 的地址为 0x8048bea, 也可以比较容易发现 phase_1 函数。

```

291 08048a56: <main>:
292 8048a56: 8d 4c 24 04      lea    0x4(%esp),%ecx
293 8048a5a: 83 e4 f0         and    $0xffffffff0,%esp
294 8048a5d: ff 71 fc         pushl  -0x4(%ecx)
295 8048a60: 55              push   %ebp
296 8048a61: 89 e5           mov     %esp,%ebp
297 8048a63: 56              push   %esi
298 8048a64: 53              push   %ebx
299 8048a65: 51              push   %ecx
300 8048a66: 83 ec 0c         sub     $0xc,%esp
301 8048a69: e8 22 ff ff ff   call    8048990 <__x86.get_pc_thunk.bx>
302 8048a6e: 81 c3 92 45 00 00 add     $0x4592,%ebx
303 8048a74: 8b 01           mov     (%ecx),%eax
304 8048a76: 8b 71 04         mov     0x4(%ecx),%esi
305 8048a79: 83 f8 01         cmp     $0x1,%eax
306 8048a7c: 0f 84 15 01 00 00 je      8048b97 <main+0x141>
307 8048a82: 83 f8 02         cmp     $0x2,%eax
308 8048a85: 0f 85 40 01 00 00 jne     8048bcb <main+0x175>
309 8048a8b: 83 ec 08         sub     $0x8,%esp
310 8048a8e: 8d 83 28 d4 ff ff lea     -0x2bd8(%ebx),%eax
311 8048a94: 50              push   %eax
312 8048a95: ff 76 04         pushl   0x4(%esi)
313 8048a98: e8 e3 fd ff ff   call    8048880 <fopen@plt>
314 8048a9d: c7 c2 c8 d3 04 08 mov     $0x804d3c8,%edx
315 8048aa3: 89 02           mov     %eax,(%edx)
316 8048aa5: 83 c4 10         add     $0x10,%esp
317 8048aa8: 85 c0           test    %eax,%eax
318 8048aaa: 0f 84 fc 00 00 00 je      8048bac <main+0x156>
319 8048ab0: e8 c1 07 00 00   call    8049276 <initialize_bomb>
320 8048ab5: 83 ec 0c         sub     $0xc,%esp
321 8048ab8: 8d 83 ac d4 ff ff lea     -0x2b54(%ebx),%eax
322 8048abe: 50              push   %eax
323 8048abf: e8 4c fd ff ff   call    8048810 <puts@plt>
324 8048ac4: 8d 83 e8 d4 ff ff lea     -0x2b18(%ebx),%eax
325 8048aca: 89 04 24         mov     %eax,(%esp)
326 8048acd: e8 3e fd ff ff   call    8048810 <puts@plt>
327 8048ad2: e8 c4 0a 00 00   call    804959b <read_line>
328 8048ad7: 89 04 24         mov     %eax,(%esp)
329 8048ada: e8 0b 01 00 00   call    8048bea <phase_1>
330 8048adf: e8 e4 0b 00 00   call    80496c8 <phase_defused>

```

图 4. 反汇编文件中的 main 函数对应的部分代码

```

397 08048bea: <phase_1>:
398 8048bea: 53              push   %ebx
399 8048beb: 83 ec 10         sub     $0x10,%esp
400 8048bee: e8 9d fd ff ff   call    8048990 <__x86.get_pc_thunk.bx>
401 8048bf3: 81 c3 0d 44 00 00 add     $0x440d,%ebx
402 8048bf9: 8d 83 64 d5 ff ff lea     -0x2a9c(%ebx),%eax
403 8048bff: 50              push   %eax
404 8048c00: ff 74 24 1c         pushl   0x1c(%esp)
405 8048c04: e8 08 06 00 00   call    8049211 <strings_not_equal>
406 8048c09: 83 c4 10         add     $0x10,%esp
407 8048c0c: 85 c0           test    %eax,%eax
408 8048c0e: 75 05           jne     8048c15 <phase_1+0x2b>
409 8048c10: 83 c4 08         add     $0x8,%esp
410 8048c13: 5b              pop     %ebx
411 8048c14: c3              ret
412 8048c15: e8 e8 08 00 00   call    8049502 <explode_bomb>
413 8048c1a: eb f4           jmp     8048c10 <phase_1+0x26>

```

图 5. 反汇编文件中 phase_1 函数对应的代码

从图 5 中可以看出 `phase_1` 调用了 `strings_not_equal`，该函数的返回结果在 `%eax` 中，`test %eax, %eax` 和 `jne 8048c15` 表示当返回结果不为 0 时，跳至 `0x8048c15`，而该处语句 `call explode_bomb`（地址为 `0x8049502`），引爆一次炸弹。猜测 `phase_1` 的功能是输入字符串，将其与预设的字符串比较，若相同则通过，若不相同则引爆炸弹。为了验证猜测是否正确，需调试运行。方法如图 6 和图 7 所示。

图 6 示意了如何获得 `phase_1` 正确的输入。

```
qinliu@ubuntu:~/ics/bomb2017202210003$ gdb bomb
GNU gdb (Ubuntu 7.7-0ubuntu3) 7.7
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from bomb...done.
(gdb) b explode_bomb
Breakpoint 1 at 0x8049502
(gdb) b phase_1
Breakpoint 2 at 0x8048bea
(gdb) b *(0x8048c04)
Breakpoint 3 at 0x8048c04
(gdb) info b
Num      Type           Disp Enb Address      What
1        breakpoint      keep y   0x08049502  <explode_bomb>
2        breakpoint      keep y   0x08048bea  <phase_1>
3        breakpoint      keep y   0x08048c04  <phase_1+26>
(gdb) r
Starting program: /home/qinliu/ics/bomb2017202210003/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
a test

Breakpoint 2, 0x08048bea in phase_1 ()
(gdb) c
Continuing.

Breakpoint 3, 0x08048c04 in phase_1 ()
(gdb) p /x $esp
$1 = 0xffffd080
(gdb) x/4xw 0xffffd080
0xffffd080: 0x0804d3e0 0x0804a564 0x00000000 0x0804d000
(gdb) p (char*) 0x0804d3e0
$2 = 0x0804d3e0 <input_strings> "a test"
(gdb) p (char*) 0x0804a564
$3 = 0x0804a564 "The future will be better tomorrow."
(gdb)
```

图 6. 获得 `phase_1` 正确输入的方法

- (1) `gdb bomb` 命令启动 `gdb`，对 `bomb` 进行调试；
- (2) 进入 `gdb` 后，`b explode_bomb` 命令对 `explode_bomb` 打断点，防止错误输入导致引爆

炸弹；

- (3) **b phase_1** 对 phase_1 函数打断点；
- (4) **b *(0x8048c04)**，对地址为 0x8048c04 的指令打断点，该指令调用 string_not_equal
- (5) **info b** 查看所有断点
- (6) **r** 运行 bomb 程序
- (7) **"a test"** 是随意输入的一个字符串
- (8) **c** 继续运行程序至下一断点
- (9) **p /x \$esp** 以 16 进制显示 esp 寄存器，得到 esp 为 0xffffd080
- (10) **x/4xw 0xffffd080** 以 16 进制显示从内存地址 0xffffd080 开始的 4 个四字节，两个四字节正是 strings_not_equal 函数的两个输出字符串首地址 0x0804d3e0 和 0x0804a564。
- (11) **p (char*) 0x804d3e0** 显示位于地址 0x804d3e0 的字符串，得到为 "a test"，这是之前随意的输入
- (12) **p (char*) 0x804a564** 显示位于地址 0x804a564 的字符串，得到为 "The future will be better tomorrow." 这便是预设字符串。

图 7 示意使用猜测的预设字符串输入通过了第一关。

```
(gdb) r
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/qinliu/ics/bomb2017202210003/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
The future will be better tomorrow.

Breakpoint 2, 0x08048bea in phase_1 ()
(gdb) c
Continuing.

Breakpoint 3, 0x08048c04 in phase_1 ()
(gdb) c
Continuing.
Phase 1 defused. How about the next one?
```

图 7. 使用猜测的预设字符串输入通过了第一关

- (1) **r** 表示运行程序，**y** 确认重头开始
- (2) 输入 **"The future will be better tomorrow."**
- (3) **c** 表示继续运行程序至下一断点
- (4) **c** 表示继续运行程序至下一断点
- (5) 显示 **"Phase 1 defused. How about the next one?"** 表示通过第一关。

另外，对于 C 源代码可显示的部分，可使用 list 命令，否则可使用 disassemble 命令显示反汇编的指令，如图 8 所示。

```

(gdb) list main
32      */
33
34      FILE *infile;
35
36      int main(int argc, char *argv[])
37      {
38          char *input;
39
40          /* Note to self: remember to port this bomb to Windows and put a
41             * fantastic GUI on it. */
(gdb) 
42
43          /* When run with no arguments, the bomb reads its input lines
44             * from standard input. */
45          if (argc == 1) {
46              infile = stdin;
47          }
48
49          /* When run with one argument <file>, the bomb reads from <file>
50             * until EOF, and then switches to standard input. Thus, as you
51             * defuse each phase, you can add its defusing string to <file> and
(gdb) disassemble phase_1
Dump of assembler code for function phase_1:
0x08048bea <+0>:    push    %ebx
0x08048beb <+1>:    sub     $0x10,%esp
0x08048bee <+4>:    call   0x8048990 <__x86.get_pc_thunk.bx>
0x08048bf3 <+9>:    add     $0x440d,%ebx
0x08048bf9 <+15>:   lea     -0x2a9c(%ebx),%eax
0x08048bff <+21>:   push    %eax
0x08048c00 <+22>:   pushl   0x1c(%esp)
0x08048c04 <+26>:   call   0x8049211 <strings_not_equal>
0x08048c09 <+31>:   add     $0x10,%esp
0x08048c0c <+34>:   test    %eax,%eax
0x08048c0e <+36>:   jne     0x8048c15 <phase_1+43>
0x08048c10 <+38>:   add     $0x8,%esp
0x08048c13 <+41>:   pop     %ebx
0x08048c14 <+42>:   ret
0x08048c15 <+43>:   call   0x8049502 <explode_bomb>
0x08048c1a <+48>:   jmp     0x8048c10 <phase_1+38>
End of assembler dump.
(gdb) 

```

图 8. gdb 中显示函数对应 c 代码和反汇编的代码

- (1) **list main** 表示显示 main 函数的源代码，默认只显示 10 行；
- (2) 输入回车键表示继续向下显示；
- (3) **disassemble phase_1** 表示对 phase_1 反汇编并显示。

图 9 显示一个炸弹引爆的示例。

```

(gdb) r
Starting program: /home/qinliu/ics/bomb2017202210003/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
stesttest
BOOM!!!
The bomb has blown up.
Your instructor has been notified.
[Inferior 1 (process 17367) exited with code 010]
(gdb) 

```

图 9. 炸弹引爆的示例

- (1) 输出错误的字符串 **steststest**;

(2) 显示

```
BOOM!!!  
The bomb has blown up.  
Your instructor has been notified.
```

还可以把输入的内容放在文件里，每一行代表一个 phase 的输入，有多少行就表示做到多少关。图 10 表示输入的内容放在 solution.txt 文件中

```
(gdb) b explode_bomb  
Breakpoint 8 at 0x8049502  
(gdb) r solution.txt  
Starting program: /home/qinliu/ics/bomb2017202210003/bomb solution.txt  
Welcome to my fiendish little bomb. You have 6 phases with  
which to blow yourself up. Have a nice day!  
  
Breakpoint 1, 0x08048bea in phase_1 ()  
(gdb) c  
Continuing.  
Phase 1 defused. How about the next one?  
  
Breakpoint 2, 0x08048c1c in phase_2 ()  
(gdb) c  
Continuing.  
That's number 2. Keep going!  
  
Breakpoint 3, 0x08048c92 in phase_3 ()  
(gdb) c  
Continuing.  
Halfway there!  
  
Breakpoint 4, 0x08048e68 in phase_4 ()  
(gdb) c  
Continuing.  
So you got that one. Try this one.  
  
Breakpoint 5, 0x08048eec in phase_5 ()  
(gdb) c  
Continuing.  
Good work! On to the next...  
  
Breakpoint 6, 0x08048f8a in phase_6 ()  
(gdb) c  
Continuing.  
Curses, you've found the secret phase!  
But finding it and solving it are quite different...  
  
Breakpoint 7, 0x080490e8 in secret_phase ()  
(gdb) c  
Continuing.  
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 18225) exited normally]  
(gdb) █
```

图 10. 把输入的内容放至文件里

特别注意：

- 不要试图去修改发送给服务器端的数据，一旦服务器端检测到错误数据，会自动记录 invalid 成绩，后续提交的结果即使正确也将无法记录。
- 不要花费精力去弄懂反汇编代码中每一条指令的作用，抓住关键线索。
- 擅于使用 gdb 去调试和查看内存和寄存器的情况（动态执行程序），而不是仅仅静态分析代码逻辑。
- 并非所有的指令和用法课堂内都会讲授，需要大家自己在网上找资料，但不必纠结于要读懂每一行指令。

4. 实验要求

- 实验截止时间：2020 年 11 月 15 日 22 时
- 该实验需提交纸质版实验报告（word 版），实验报告中需说明每一关的具体解决过程，类似于本实验说明对破解 phase_1 的描述过程，实验报告截止时间是 2020 年 11 月 18 日 24 时提交至课代表（不要通过邮件发送，请等待课代表发起群作业），名称：学号_姓名_BombLab 实验报告。
- 实验报告中写清每一关的破解思路和过程（类似实验说明中的示例）。如果仅凭破解工具而无反汇编级分析过程，即使通过服务器检查，后期也会判定为不合格。
- 实验报告模板中提供了标题和目录的格式，撰写时请根据需要增补。