

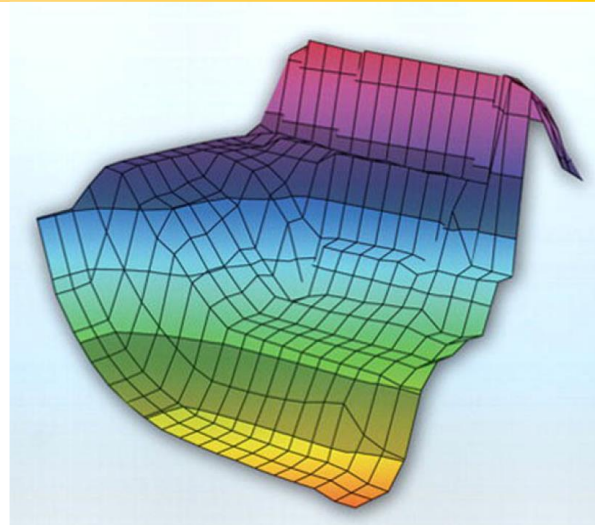
# 《计算机系统基础实验》

## Lab2 Binary Bombs

2023 春季

华中科技大学

2023-05-25



- 本实验中，你要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”（二进制炸弹，简称炸弹）是一个Linux可执行C程序，包含phase1~phase6共6个阶段。
- 炸弹运行的每个阶段要求你输入一个**特定的字符串**，若你的输入符合程序预期的输入，该阶段的炸弹就被“拆除”，否则炸弹“爆炸”并打印输出“BOOM!!!”字样。
- 实验的目标是你药拆除尽可能多的炸弹阶段。

提示：每个炸弹阶段考察了机器级语言程序的一个不同方面，  
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：含switch语句
- 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，但只有当你在第4阶段的解之后附加一特定字符串后才会出现。

- 拆弹技术：为了完成二进制炸弹拆除任务，你需要
  - ① 使用gdb调试器和objdump来反汇编炸弹的可执行文件；
  - ② 单步跟踪调试每一阶段的机器代码
  - ③ 理解每一汇编语言代码的行为或作用，
  - ④ 进而设法“推断”出拆除炸弹所需的目标字符串。
  - ⑤ 这可能需要你在每一阶段的开始代码前和引爆炸弹的函数前设置断点，以便于调试。
- 实验语言：C语言，实验环境：linux

本次实验中，每位同学会得到一个不同的binary bomb二进制可执行程序及其相关文件，其中包含如下文件：

- bomb: bomb的可执行程序。
- bomb.c: bomb程序的main函数。
- ID
- README

用文本编辑器打开看看就知道里面有什么了

- bomb: 是一个linux下可执行程序，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，期待你按行输入每一阶段用来拆除炸弹的字符串，并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。
- bomb.c: bomb的主程序，但不是全部，这里面你看不到炸弹

- 1) 可以在命令行运行bomb，然后根据提示，逐阶段输入拆弹字符串（见演示）。
- 2) 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，如ans.txt，然后作为运行程序时的命令行参数传给程序。
  - ◆ 结果文件格式：每个拆弹字符串一行，最多7行（包含最后特殊阶段），除此之外不要包含任何其它字符。

范例如下：

```
string1  
string2  
.....  
string6  
string7
```

◆ 使用方法: `./bomb ans .txt`

- 程序会自动读取文本文件中的字符串，并依次检查对应每一阶段的字符串来决定炸弹拆除成败。

◆ 本次实验需要提交的结果包括：实验报告和结果文件

□ 结果文件：即上述的ans.txt，重新命名如下：

班级\_学号.txt，如CS1201\_U201214795.txt

信安 IS 物联网 IT 计算机 CS 卓越班 ZY ACM班 ACM

□ 实验报告：Word文档。在实验报告中，对你拆除了炸弹的每一道题，用文字详细描述分析求解过程。

排版要求：字体：宋体；字号：标题三号，正文小四正文；

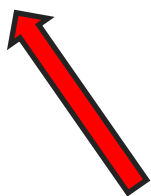
行间距：1.5倍；首行缩进2个汉字；程序排版要规整

□ 以班为单位集中打包发送至hkliu@hust.edu.cn



## ■ 直接运行bomb

```
BOMB: Command not found  
acd@ubuntu:~/Lab1-3/bomblab/CS201401/U201414557$ ./bomb  
Welcome to my fiendish little bomb. You have 6 phases with  
which to blow yourself up. Have a nice day!
```



在这个位置初入阶段1的拆弹密码，如： *This is a nice day.*

## ■ 你的工作：猜这个密码？

□ 下面以phase1为例介绍一下基本的实验步骤：

## 第一步：调用

**objdump -d bomb > asm.txt**

对bomb进行反汇编并将汇编代码输出到asm.txt中。

## 第二步：查看汇编源代码asm.txt文件

首先，查找“main”，找到main函数的位置

然后，在main函数中找到如下语句（这里为phase1函数在main()函数中被调用的位置）：

8048a4c:	c7 04 24 01 00 00 00	movl \$0x1,(%esp)
8048a53:	e8 2c fd ff ff	call 8048784 <__printf_chk@plt>
8048a58:	e8 49 07 00 00	call 80491a6 <read_line>
8048a5d:	89 04 24	mov %eax,(%esp)
8048a60:	e8 a1 04 00 00	call 8048f06 <phase_1>
8048a65:	e8 4a 05 00 00	call 8048fb4 <phase_defused>
8048a6a:	c7 44 24 04 40 a0 04	movl \$0x804a040,0x4(%esp)

# 实验步骤演示 (续)

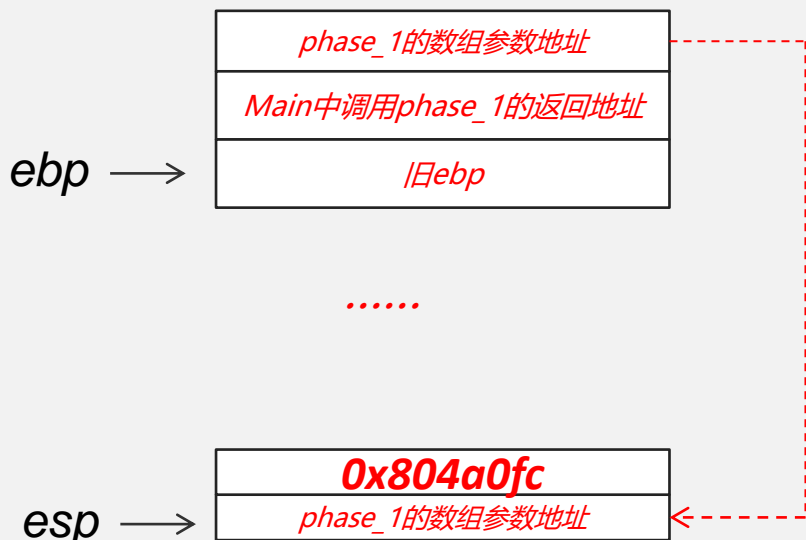
**第三步：**在反汇编文件中继续查找phase\_1的位置，如：

08048f06 <phase\_1>:

8048f06: 55  
8048f07: 89 e5  
8048f09: 83 ec 18  
8048f0c: c7 44 24 04 fc a0 04  
8048f13: 08

```
push %ebp  
mov %esp,%ebp  
sub $0x18,%esp  
movl $0x804a0fc,0x4(%esp)
```

```
mov 0x8(%ebp),%eax  
mov %eax,(%esp)  
call 8048f4b <strings_not_equal>  
test %eax,%eax  
je 8048f28 <phase_1+0x22>  
call 8049071 <explode_bomb>  
leave  
ret
```



从上面的语句中可以看出<strings\_not\_equal>所需要的两个变量是存在于%esp所指向的堆栈存储单元里。

**第四步：**，在main()函数的汇编代码中，可以进一步找到：

```
8048a58: e8 49 07 00 00      call    80491a6 <read_line>
8048a5d: 89 04 24            mov     %eax,(%esp)
```

这两条语句告诉我们%eax里存储的是调用read\_line()函数后返回的结果，也就是用户输入的字符串首地址，所以可以很容易地推断出和用户输入字符串相比较的字符串的存储地址为0x804a0fc，因为调用strings\_not\_equal前有语句：

```
8048f0c: c7 44 24 04 fc a0 04    movl    $0x804a0fc,0x4(%esp)
```

也许你看到的程序和前面的不一样，而是这样的：

```
08048b90 <phase_1>:
8048b90:      83 ec 1c          sub     $0x1c,%esp
8048b93:      c7 44 24 04 44 a1 04  movl    $0x804a144,0x4(%esp)
8048b9a:      08
8048b9b:      8b 44 24 20       mov     0x20(%esp),%eax
8048b9f:      89 04 24          mov     %eax,(%esp)
8048ba2:      e8 73 04 00 00    call   804901a <strings_not_equal>
8048ba7:      85 c0            test    %eax,%eax
8048ba9:      74 05            je      8048bb0 <phase_1+0x20>
8048bab:      e8 75 05 00 00    call   8049125 <explode_bomb>
8048bb0:      83 c4 1c          add     $0x1c,%esp
8048bb3:      c3              ret
```

- ◆ 现在的gcc可以不使用ebp寄存器了。这样在程序就不需要保存、修改、恢复ebp。而这样ebp也就是一个free的寄存器而在函数中用作它用。

0x804a0fc里存放是是什么呢？

下面使用gdb查看这个地址存储的数据内容。具体过程如下：

**第五步：执行：./bomb/bomblab/src\$ gdb bomb, 显示如下：**

```
GNU gdb (GDB) 7.2-ubuntu
```

```
Copyright (C) 2010 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 "i686-linux-gnu".
```

```
For bug reporting instructions, please see:
```

```
<http://www.gnu.org/software/gdb/bugs/>...
```

```
./bomb/bomblab/src/bomb...done.
```

```
(gdb)
```

然后执行以下操作：

(gdb) **b main**

在main函数的开始处设置断点

Breakpoint 1 at 0x80489a5: file bomb.c, line 45.

(gdb) **r**

从gdb里运行bomb程序

Starting program: ./bomb/bomblab/src/bomb

Breakpoint 1, main (argc=1, argv=0xbffff3f4) at bomb.c:45

45           if (argc == 1) {

运行后，暂停在断点1处

(gdb) **ni**   单步执行机器指令

0x080489a8   45           if (argc == 1) {

(gdb) **ni**

这里可以看到执行到哪一条C语句

46           infile = stdin;

(gdb) **ni**

# 实验步骤演示 (续)

```
0x080489af      46                infile = stdin;
(gdb) ni
0x080489b4      46                infile = stdin;
(gdb) ni
67              initialize_bomb();
(gdb) ni
printf (argc=1, argv=0xbffff3f4) at /usr/include/bits/stdio2.h:105
105             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
(gdb) ni
0x08048a38      105             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
(gdb) ni
0x08048a3f      105             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
(gdb) ni
Welcome to my fiendish little bomb. You have 6 phases with
0x08048a44 in printf (argc=1, argv=0xbffff3f4)
at /usr/include/bits/stdio2.h:105
105             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
(gdb) ni
0x08048a4c      105             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
(gdb) ni
0x08048a53      105             return __printf_chk (__USE_FORTIFY_LEVEL - 1, __fmt, __va_arg_pack ());
(gdb) ni
which to blow yourself up. Have a nice day!
main (argc=1, argv=0xbffff3f4) at bomb.c:73
```

一直ni下去，直到下面的语句：

```
73          input = read_line();                /* Get input */
(gdb) ni          /*如果是命令行输入，这里输入你的拆弹字符串*/
74          phase_1(input);                      /* Run the phase */
```

在这个位置查看地址0x804a0fc处的内容：



```
(gdb) x/20x 0x804a0fc
```

0x804a0fc:	0x6d612049	0x73756a20	0x20612074	0x656e6572
0x804a10c:	0x65646167	0x636f6820	0x2079656b	0x2e6d6f6d
0x804a11c:	0x00000000	0x08048eb3	0x08048eac	0x08048eba
0x804a12c:	0x08048ec2	0x08048ec9	0x08048ed2	0x08048ed9
0x804a13c:	0x08048ee2	0x0000000a	0x00000002	0x0000000e

```
(gdb)
```

问，上面字节是什么内容呢？

从地址0x804a0fc开始到“0x00”字节结束（C语言字符串数据的结束符）的字节序列就是拆弹字符串ASCII码，根据低位存储规则，查表即得该字符串为“I am just a renegade hockey mom.”，从而完成了第一个密码的破译。

# 实验步骤演示 (续)

(gdb) x/20x 0x804a0fc

0x804a0fc:	0x6d612049	0x73756a20	0x20612074	0x656e6572
	m a l	s u j	a t	e n e r
0x804a10c:	0x65646167	0x636f6820	0x2079656b	0x2e6d6f6d
	e d a g	c o h	y e k	. m o m
0x804a11c:	0x00000000	0x08048eb3	0x08048eac	0x08048eba
	0			
0x804a12c:	0x08048ec2	0x08048ec9	0x08048ed2	0x08048ed9
0x804a13c:	0x08048ee2	0x0000000a	0x00000002	0x0000000e

(gdb)

I am just a renegade hockey mom.", 从而完成了第一个密码的破译。

正确拆弹的另一个实例的显示（阶段1）：

```
acd@ubuntu:~/Lab1-3/bomblab/src$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
You can Russia from land here in Alaska.
Phase 1 defused. How about the next one?
```

拆弹失败的显示（阶段1）：

```
acd@ubuntu:~/Lab1-3/bomblab/src$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
You can russia from land here in Alaska.

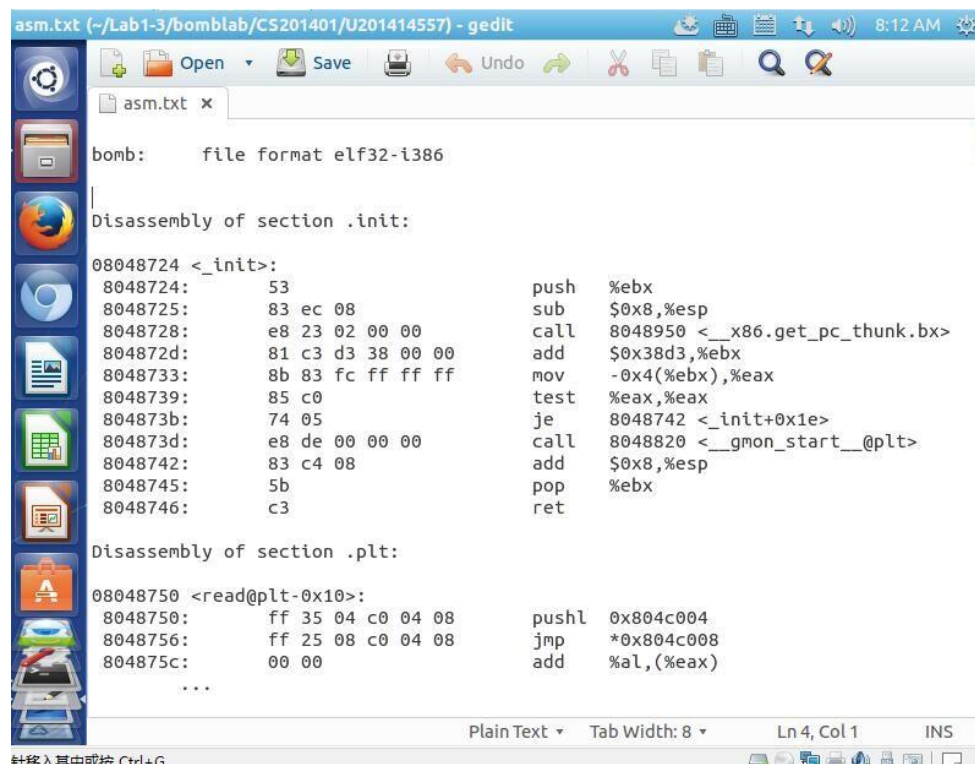
BOOM!!!
The bomb has blown up.
```

## 1) 使用objdump 反汇编bomb的汇编源程序

objdump -d bomb > asm.txt

">" :重定向, 将反汇编出来的源程序输出至文件asm.txt中

## 2) 查看反汇编源代码: gedit asm.txt



```
asm.txt (~/.Lab1-3/bomblab/CS201401/U201414557) - gedit
bomb:      file format elf32-i386

Disassembly of section .init:

08048724 <_init>:
8048724:  53                push    %ebx
8048725:  83 ec 08          sub     $0x8,%esp
8048728:  e8 23 02 00 00    call   8048950 <__x86.get_pc_thunk.bx>
804872d:  81 c3 d3 38 00 00 add     $0x38d3,%ebx
8048733:  8b 83 fc ff ff ff mov     -0x4(%ebx),%eax
8048739:  85 c0             test    %eax,%eax
804873b:  74 05             je      8048742 <_init+0x1e>
804873d:  e8 de 00 00 00    call   8048820 <__gmon_start__@plt>
8048742:  83 c4 08          add     $0x8,%esp
8048745:  5b               pop     %ebx
8048746:  c3               ret

Disassembly of section .plt:

08048750 <read@plt-0x10>:
8048750:  ff 35 04 c0 04 08 pushl   0x804c004
8048756:  ff 25 08 c0 04 08 jmp     *0x804c008
804875c:  00 00             add     %al,(%eax)
...
```

如何在asm定位main或  
phase\_1等符号?  
find查找相应字符串即可

## 3) gdb的使用

调试bomb: `./bomb/bomblab/src$ gdb bomb`

## 4) gdb常用指令

**l**: (list) 显式当前行的上、下若干行C语句的内容

**b**: (breakpoint) 设置断点

- ◆ 在main函数前设置断点: **b main**

- ◆ 在第5行程序前设置断点: **b 5**

**r**: (run)执行, 直到第一个断点处, 若没有断点, 就一直执行下去直至结束。

**ni/stepi**: (next/step instructor) 单步执行机器指令

**n/step**: (next/step) 单步执行C语句

**x**: 显示内存内容

基本用法: 以十六进制的形式显示0x804a0fc处开始的20个字节的內容:

**(gdb) x/20x 0x804a0fc**

■ 0x804a0fc:	<b>0x6d612049</b>	<b>0x73756a20</b>	<b>0x20612074</b>	<b>0x656e6572</b>
■ 0x804a10c:	<b>0x65646167</b>	<b>0x636f6820</b>	<b>0x2079656b</b>	<b>0x2e6d6f6d</b>
■ 0x804a11c:	<b>0x00000000</b>	0x08048eb3	0x08048eac	0x08048eba
■ 0x804a12c:	0x08048ec2	0x08048ec9	0x08048ed2	0x08048ed9
■ 0x804a13c:	0x08048ee2	0x0000000a	0x00000002	0x0000000e

**q**: 退出gdb, 返回linux

gdb其他命令的用法详见使用手册, 或联机help