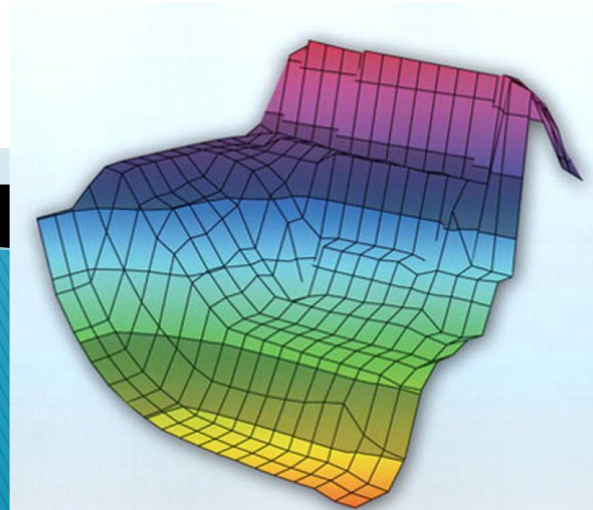


Lab2 Binary Bombs



实验概述

- ▶ 使用课程知识拆除一个“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理解。
- ▶ 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。
- ▶ 炸弹运行各阶段要求输入一个字符串，若输入符合程序预期，该阶段炸弹被“拆除”，否则“爆炸”。
- ▶ 实验目标是你需要拆除尽可能多的炸弹。

实验概述

- ▶ 每个炸弹阶段考察**机器级语言程序**不同方面，难度递增
 - 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
 - 隐藏阶段，第4阶段的之后附加一特定字符串后才会出现

实验技能

- ▶ 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；
 - ② 单步跟踪调试每一阶段的机器代码
 - ③ 理解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 需要在每一阶段的开始代码前和引爆炸弹的函数前设置断点，便于调试。
- ▶ 实验语言：C语言，实验环境：linux

文件说明

- ▶ 炸弹文件包：

- 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）。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主程序，不是全部，看不到炸弹

实验结果及结果文件

- ▶ 可在命令行运行bomb，然后根据提示，逐阶段输入拆弹字符串（见演示）。
- ▶ 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solution.txt，然后作为参数传给程序。
- ▶ 结果文件格式：每个拆弹字符串一行，**回车结束**，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
```

实验结果文件

- ▶ 使用方法： `./bomb solution.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。

实验报告和结果文件

- ▶ 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：将上述的solution.txt重新命名为

学号.txt

- 实验报告：学号 - 姓名 - Lab1.pdf：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，并对实验结果截屏。排版要求：使用PA报告模板（很多人的PA报告看到脑袋爆炸！）
- 提交时间：2019/5/28 18:00:00（以提交网站为准）

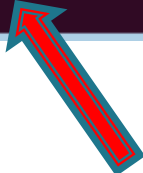
注意事项

- ▶ 提交的报告中需对自己的心路历程和实验步骤进行相应描述，不能空有答案
- ▶ 这也就要求我们尽量能够一边做一边写报告，不要像PA那样写完了再写报告，然后连做了什么都不知道
- ▶ 会有人使用各类其他工具完成实验。**不管使用什么工具，最终要落实到使用GDB进行程序调试，因为实验的目的是练习GDB调试。**报告一定是以使用GDB，并读懂反汇编为基础撰写的。

实验步骤提示

- ▶ 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：

实验步骤演示

第一步： **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，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）

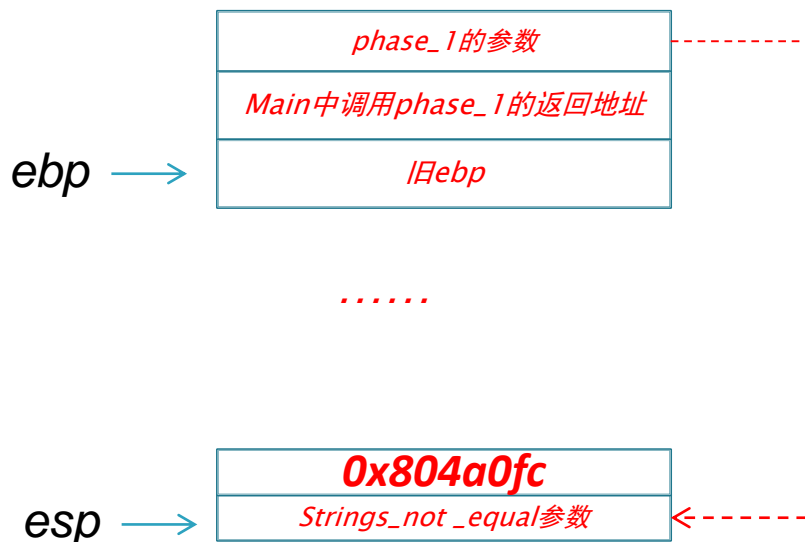
第三步：在反汇编文件中继续查找phase_1的位置，如：

08048f06 <phase_1>:

8048f06: 55
8048f07: 89 e5
8048f09: 83 ec 18
8048f0c: c7 44 24 04 fc a0 04
8048f13: 08
8048f14: 8b 45 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
```



<strings_not_equal>两个变量存在于%esp所指向的堆栈存储单元里。

实验步骤演示（续）

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
```

◆ gcc可以不使用ebp，程序不需要保存、修改、恢复ebp。这样ebp也可以当通用寄存器使用

实验步骤演示（续）

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)
```

%eax里存储的是调用read_line()函数返回值，也是用户输入的字符串首地址，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)
```

实验步骤演示（续）

0x804a0fc里存放是是什么呢？

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：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)

实验步骤演示（续）

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：

实验步骤演示（续）

(gdb) x/2s 0x804a0fc

0x804a0fc: "I am just a renegade hockey mom "

0x804a132: ""

(gdb) Quit

(gdb)

显示地址0x804a0fc开始的2个字符串

“I am just a renegade hockey mom.”就是第一个密码

另外一种方法

- ▶ Objdump --start-address=**0x804a0fc -s bomb**

正确拆弹的另一个实例的显示（阶段1）：

```
The bomb has blown up.  
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.
```

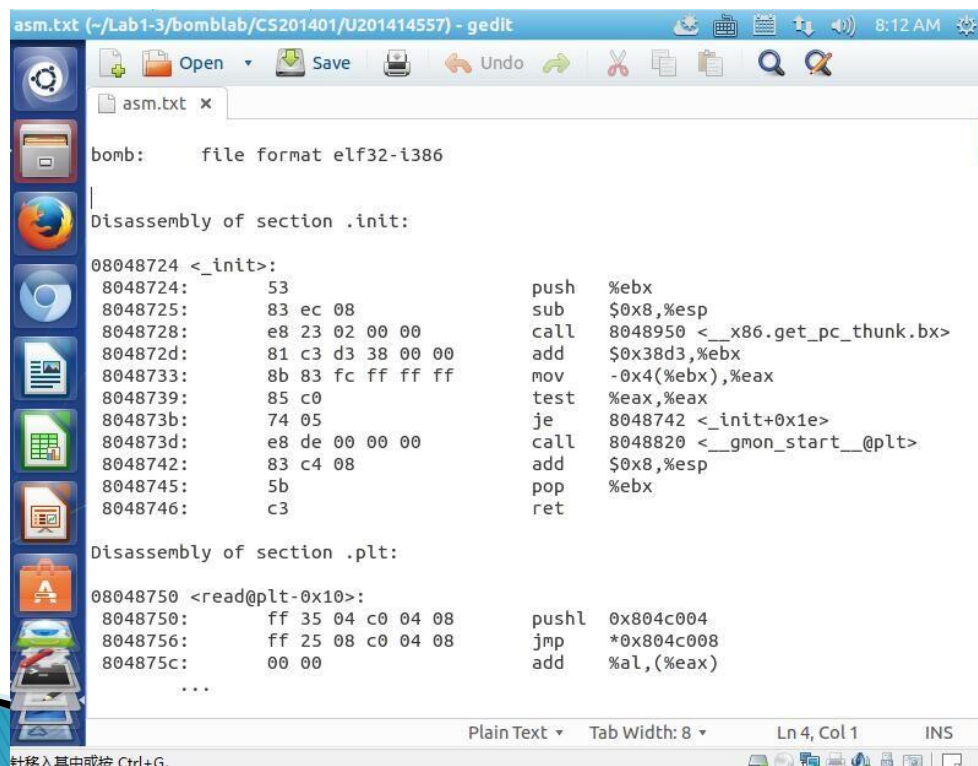
Gdb和objdump的使用

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: vim 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   08048950 <_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      08048742 <_init+0x1e>
804873d:      e8 de 00 00 00    call   08048820 <__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

Gdb和objdump的使用

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

Gdb和objdump的使用

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:	0x6d612049	0x73756a20	0x20612074	0x656e6572
▶ 0x804a10c:	0x65646167	0x636f6820	0x2079656b	0x2e6d6f6d
▶ 0x804a11c:	0x00000000	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