LAB2 实验指导

概述

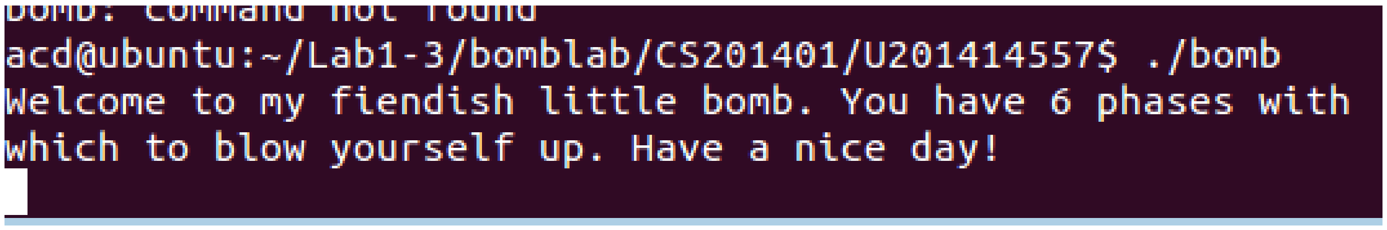
- 每个学生一个不同的Linux可执行文件（二进制炸弹)
  - 包括六个阶段和一个秘密阶段
  - 各阶段要求用户输入一个字符串
  - 如字符串满足程序要求，炸弹拆除，否则引爆
- 拆弹过程
  - 反汇编，gdb 动态跟踪
  - 理解二进制程序功能
  - 推测拆弹密码

Lab2 Binary Bombs 实验介绍

- 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阶段之后附加特定字符串后出现

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?**

学生拆弹

- ./bomb <solution.txt

  Lab2实验系统会自动上传拆弹记录，无需特殊参数

```
./bomb <solution.txt
    Welcome to my fiendish little bomb. You have 6 phases with
    which to blow yourself up. Have a nice day!
    Phase 1 defused. How about the next one?
    That‘s number 2.  Keep going!
    Halfway there!
    So you got that one.  Try this one.
    Good work!  On to the next...
    Curses, you’ve found the secret phase!
    But finding it and solving it are quite different...
    Wow! You‘ve defused the secret stage!
    Congratulations! You’ve defused the bomb!
    Your instructor has been notified and will verify your solution.    自动上传拆弹记录
```

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                      push   %ebp
8048f07:        89 e5                   mov    %esp,%ebp
8048f09:        83 ec 18                sub    $0x18,%esp
8048f0c:        c7 44 24 04 fc a0 04    movl   $0x804a0fc,0x4(%esp)
8048f13:        08
                                        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
```

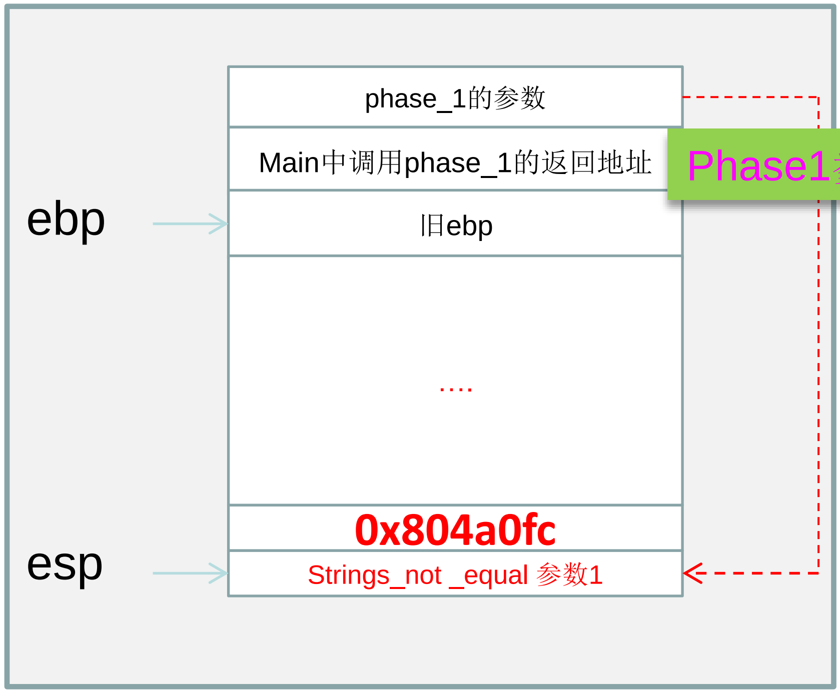
数据区地址

参数2

参数1

Phase1参数

判断是否成功



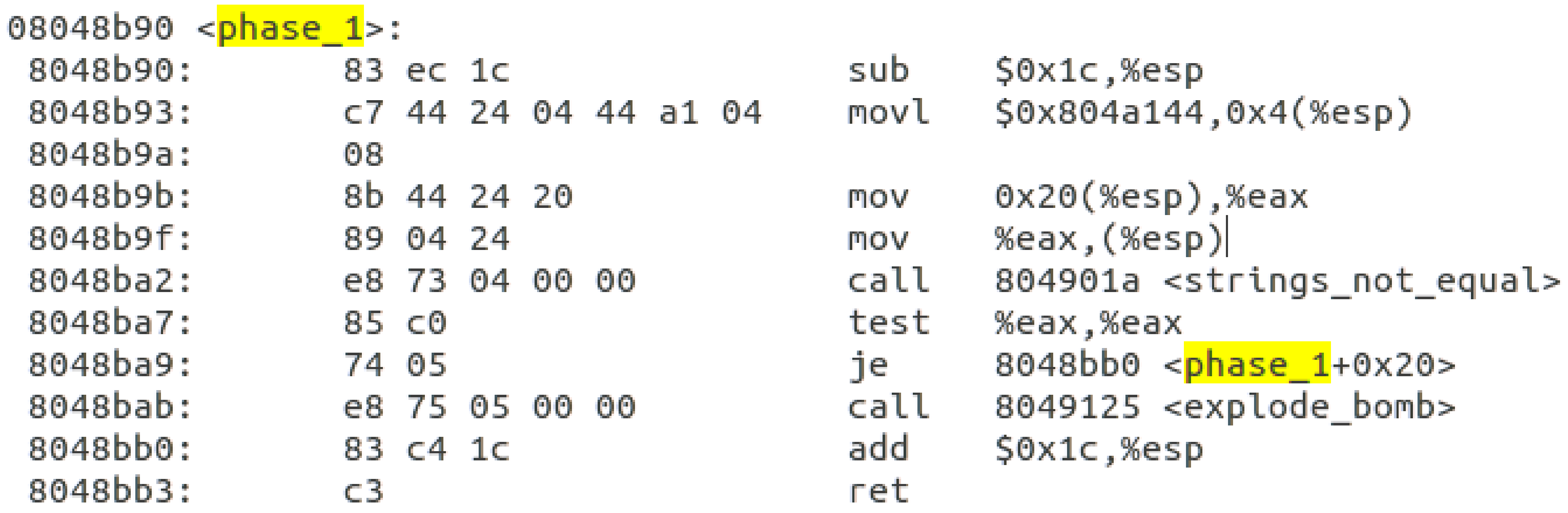
<strings_not_equal>函数两个参数
存在于%esp所指向的堆栈存储单元里。

ASCII码/ ISO-646-US标准

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0																
1																
2	20	21 !	22 "	23 #	24 $	25 %	26 &	27 '	28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
3	30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7	38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
4	40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G	48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
5	50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W	58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
6	60 `	61 a	62 b	63 c	64 d	65 e	66 f	67 g	68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
7	70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w	78 x	79 y	7A z	7B {	7C |	7D }	7E ~	

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，推测拆弹密码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)
```

phase 3:switch-case语句举例

```
int sw_test(int a, int b, int c)
{
  int result;
  switch(a) {
  case 15:
     c=b&0x0f;
  case 10:
     result=c+50;
     break;
  case 12:
  case 17:
     result=b+50;
     break;
  case 14:
     result=b
     break;
  default:
     result=a;
  }
  return result;
}
```

```
  movl  8(%ebp), %eax
  subl  $10, %eax
  cmpl  $7, %eax
  ja    .L5
  jmp    *.L8( , %eax, 4)
.L1:
  movl  12(%ebp), %eax
  andl  $15, %eax
  movl  %eax, 16(%ebp)
.L2:
  movl  16(%ebp), %eax
  addl     $50, %eax
  jmp  .L7
.L3:
  movl  12(%ebp), %eax
  addl     $50, %eax
   jmp    .L7
.L4:
   movl  12(%ebp), %eax
   jmp  .L7
.L5:
   addl  $10, %eax
.L7:
```

R[eax]=a-10=i

if (a-10)>7 转 L5

转.L8+4*i 处的地址

跳转表在目标文件的只读节中，按4字节边界对齐。

```
   .section   .rodata
   .align  4
.L8                     a=
   .long    .L2         10
   .long    .L5         11
   .long    .L3         12
   .long    .L5         13
   .long    .L4         14
   .long    .L1         15
   .long    .L5         16
   .long    .L3         17
```

Gdb调试

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）

```
(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
                    # 运行后，暂停在断点1处
Breakpoint 1, main (argc=1, argv=0xbffff3f4) at bomb.c:45
45          if (argc == 1) {
(gdb) ni                 #单步执行机器指令
0x080489a8    45          if (argc == 1) {
(gdb) ni
46                infile = stdin;   #这里可以看到执行到哪一条C语句
(gdb) ni
```

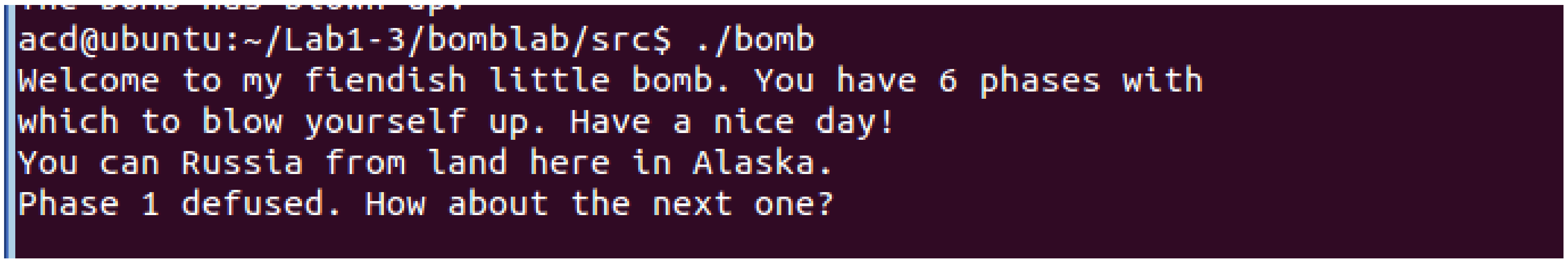
实验步骤演示（续）

```
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              */
(gdb) x/2s 0x804a0fc     #查看地址0x804a0fc处两个字符串：
0x804a0fc:      " I am just a renegade hockey mom "
0x804a132:      ""
(gdb) q                         #退出gdb
Objdump --start-address=0x804a0fc –s bomb   #方法2
```

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

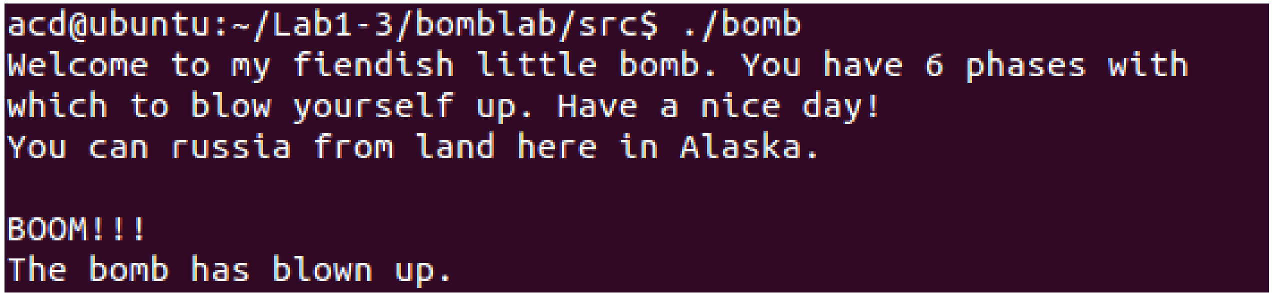
拆弹现场演示

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.
```

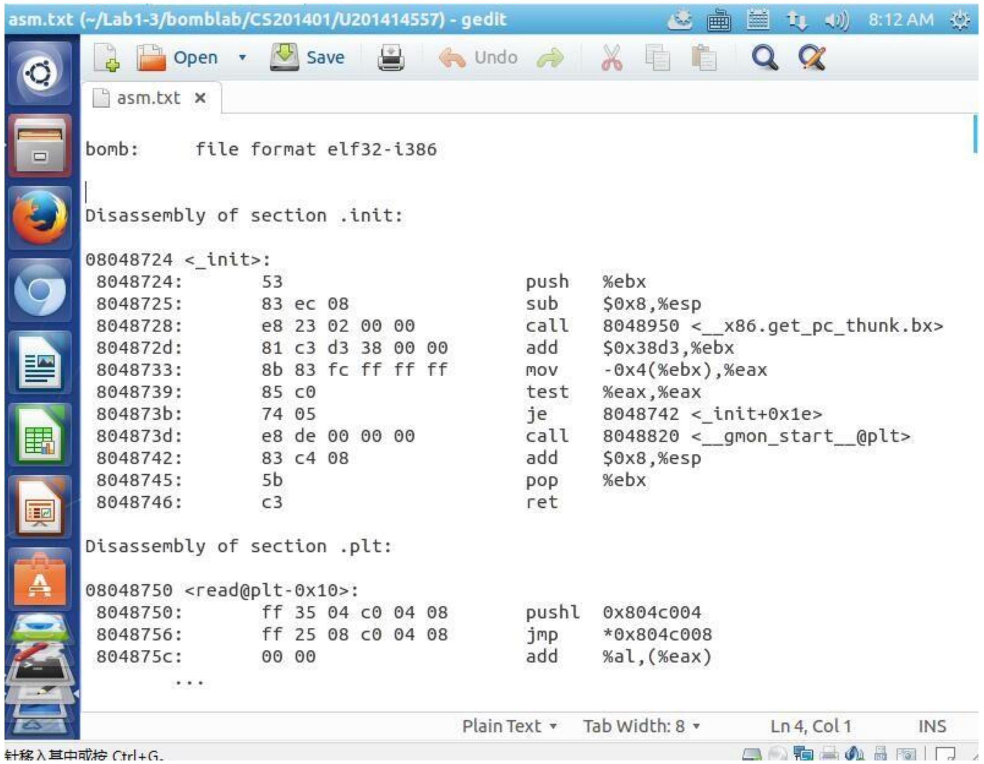
Gdb和objdump的使用

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：gedit asm.txt



```
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

Gdb和objdump的使用

3）gdb的使用

$ 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式0x804a0fc处开始的20个字节的内容：

(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
```

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