LAB2：二进制炸弹

请在实验截止前务必确认提交的实验结果符合实验说明中的提交要求(命名、格式等)，建议在提交后下载提交的文件进行确认。如果由于提交不符合实验要求而造成评分程序扣分，责任自负。

简介

在本实验中，你需要使用课程所学知识拆除一个“binary bombs”，从而加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知识的理解和掌握。 一个“binary bombs”（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）以及1个隐藏阶段。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!!!”并转到下一阶段等待你的输入。实验的目标是拆除尽可能多阶段的炸弹。

每个炸弹阶段考察了机器级程序语言的一个不同方面：

- 阶段0：浮点表示
- 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

隐藏阶段只有当你在阶段4的拆解字符串后再附加一特定字符串后才会出现（作为最后一个阶段）。

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单步跟踪每一阶段的机器代码（例如可在每一阶段的开始代码前和引爆炸弹的函数前设置断点），从中理解每一汇编语言代码的行为和作用，进而设法推断拆除炸弹所需的目标字符串。

实验语言：C, 汇编；实验环境：Linux i386（32bits）

实验数据

在本实验中，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，可从(http://114.212.86.190/course/ics17a/getbomb.htm)下载，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。

运行./bomb可执行程序时可使用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引爆该阶段的炸弹。

提示：在拆除各阶段炸弹的过程中，你可以选择跳过一些阶段（例如暂时未能拆除的阶段）以按任意顺序尝试各阶段和测试特定阶段的拆除字符串正确与否。为此，你可以在要跳过的阶段中输入任意的非空字符串（从而引爆相应阶段的炸弹——注意程序不会中止）从而快速跳过该阶段。

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（就像你需要提交的实验结果文件，见下说明），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。

实验提交要求

- **提交文件名：学号.txt**
- **提交文件格式：每个拆除字符串一行，除此之外不要包含任何其它字符，范例如下：**

```
string0
string1
string2
string3
...
...
...
...
```

注意：

1. 提交文件必须采用Unix文本格式（换行字符不同于Windows格式），建议在实验所用的LInux环境中编辑生成该提交文件。另外，注意最后的字符串后也要进行换行（即所有字符串必须以换行结尾）。
2. 如果你未能完成某阶段，务必用任一非空字符串作为该阶段的拆除字符串，并放置于如上所示提交文件的相应行中，即不能放置一空行或直接省略该行。否则，你后续阶段的拆除字符串会被用于该缺少拆除字符串的阶段，造成所有后续阶段出错。

检查提交结果（强烈建议在提交前、后均进行检查）

方法：将提交文件作为bomb程序的唯一命令行参数：

```
./bomb 学号.txt
```

程序将依次检查每一阶段拆除字符串的正确性，并仅在提交结果全部正确时，最后输出“Congratulations! You've defused the bomb!”（注意：如果漏掉了隐藏阶段但其余阶段都正确的话也会如此输出，但计分上不算完成了所有阶段）。否则，程序将在每个发生错误的阶段处输出炸弹爆炸的提示信息后，继续进行后续阶段的检查，实验总分将依据正确完成的阶段个数来计算。
注意：实验评分程序同样使用此方式检查你实验结果的正确性。

实验工具

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主要帮忙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）：

1. 装载、启动被调试的程序。
2. 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。
3. 动态改变程序的执行环境，如修改变量的值。

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。

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。

strings

该命令可以显示二进制程序中的所有可打印字符串。

实验步骤提示

下面以“阶段1”为例介绍一下基本的实验步骤： 首先调用“objdump –d bomb > disassemble.txt”对bomb进行反汇编并将汇编源代码输出到“disassemble.txt”文本文件中。查看该汇编源代码文件，我们可以在main函数中找到如下语句，从而得知phase1的处理程序包含在“main()“函数所调用的函数“phase_1()”中：

```
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)
```

接下来，我们在反汇编文件中继续查找phase_1的具体定义，如下所示：

```
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
8048f14:       8b 45 08                mov    0x8(%ebp),%eax
8048f17:       89 04 24                mov    %eax,(%esp)
8048f1a:       e8 2c 00 00 00          call   8048f4b <strings_not_equal>
8048f1f:       85 c0                   test   %eax,%eax
8048f21:       74 05                   je     8048f28 <phase_1+0x22>
8048f23:       e8 49 01 00 00          call   8049071 <explode_bomb>
8048f28:       c9                      leave
8048f29:       c3                      ret
8048f2a:       90                      nop
8048f2b:       90                      nop
8048f2c:       90                      nop
8048f2d:       90                      nop
8048f2e:       90                      nop
8048f2f:       90                      nop
```

从上面的语句中我们可以看出<strings_not_equal>所需要的两个变量是存在%esp所指向的堆栈存储单元里。从前面的main()函数中，我们可以找到

```
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()函数返回的结果，也就是用户输入的字符串，所以我们很容易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，因此我们可以使用gdb查看这个地址存储的数据内容，具体过程如下：

```
./bomb/bomblab/src$ gdb bomb
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) b main
Breakpoint 1 at 0x80489a5: file bomb.c, line 45.
(gdb) r
```

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

Breakpoint 1, main (argc=1, argv=0xbffff3f4) at bomb.c:45
45          if (argc == 1) {
(gdb) ni
0x080489a8      45          if (argc == 1) {
(gdb) ni
46              infile = stdin;
(gdb) ni
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
73          input = read_line();             /* Get input                   */
(gdb) ni
74          phase_1(input);                  /* Run the phase               */
(gdb) x/40x 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
```

```
0x804a14c <array.3474+12>:      0x00000007      0x00000008      0x0000000c      0x0000000f
0x804a15c <array.3474+28>:      0x0000000b      0x00000000      0x00000004      0x00000001
0x804a16c <array.3474+44>:      0x0000000d      0x00000003      0x00000009      0x00000006
0x804a17c <array.3474+60>:      0x00000005      0x25206425      0x73252064      0x45724400
0x804a18c:      0x006c6976      0x4f4f420a      0x2121214d      0x6854000a
(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)
```

其中从0x804a0fc地址开始到“0x00”字节结束（记得C语言字符串数据的表示要求？）的字节序列就是字符串的ASCII码，根据低位存储规则，我们可以查表得到该字符串为”I am just a renegade hockey mom.“从而完成了第一个密码的破译。