

Lab2: Binary Bomb 实验指导学生书

1. 实验简介

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，下文将简称为炸弹）是一个 Linux 可执行 C 程序，包含了 6 个阶段（phase1~phase6）。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。实验的目标是拆除尽可能多的炸弹层次。

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
- * 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。

为了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”出拆除炸弹所需的目标字符串。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

2. 实验文件

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。

运行./bomb: **./bomb** 是一个可执行程序，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。

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（比如：**result.txt**），然后作为运行程序时的唯一的一个命令行参数传给程序（**./bomb result.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。

3. 实验提交要求

提交文件名: **学号.txt**（也就是上面的 **result.txt**，需要用你的学号改个名称）

提交文件格式: 每个拆弹字符串一行，除此之外不要包含任何其它字符。

范例如下。

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

4. 实验工具

本实验所需使用的一些软件工具和用法：

1) 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** 来帮助你对程序进行分析。

GDB 是 **GNU** 开源组织发布的一个强大的交互式程序调试工具。**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。

2) objdump 反汇编工具

➤ **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>
```

所以，你还需要 **gdb** 来帮助你确定这个语句的具体功能。

3) strings

该命令可以显示二进制程序中的所有可打印字符串。

5. 实验步骤提示

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 > 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。

第二步：查看该汇编源代码文件 **disassemble.txt**，你可以在 **main** 函数中找到如下语句（通过查找 “**phase_1**”），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)

第三步：在反汇编文件中继续查找 phase_1，找到其具体定义的位置，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于%esp 所指向的堆栈存储单元里（strings_not_equal 是程序中已经设计好的一个用于判断两个字符串是否相等的函数）。

第四步：，从前面的 main()函数中，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：

在 phase1 的反汇编代码中有语句：

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

指出了 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：

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) 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 /*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/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。

- 注意：本实验需要撰写并打印实验报告，请在规定的时间内提交，电子档按“班级_学号_姓名”命名提交。