Bomblab 实验操作指导

申请炸弹文件

打开实验主页，进入 http://ecop.jinhangdev.cn/bomblab-getbomb/ ，将看到 CS:APP Binary Bomb Request 的页面，在 User name 处输入自己的学号，电子邮件地址可以随意填写。填写完成后点击 `Submit` ，稍等片刻浏览器将弹出下载框，将 `bomb_id.tar` 保存到计算机中。

传输到虚拟机

先运行虚拟机，然后使用 FileZilla 连接到虚拟机，将 `bomb_id.tar` 传输到虚拟机中并解压：

```
sysu@debian:~$ ls
bomb_jinhang.tar  ex  ex.o  ex.S  folder
sysu@debian:~$ tar -xvf bomb_jinhang.tar
bomb1/README
bomb1/bomb.c
bomb1/bomb
sysu@debian:~$ ls
bomb1  bomb_jinhang.tar  ex  ex.o  ex.S  folder
sysu@debian:~$
```

这里出现了一个新的目录 `bomb1` ，这个就是你的炸弹编号，记住这个编号，后面还会用来查看排行榜。

开始实验

进入 `bomb2` ，里面有三个文件：

```
sysu@debian:~$ cd bomb1/
sysu@debian:~/bomb1$ ls
bomb  bomb.c  README
sysu@debian:~/bomb1$
```

- bomb：炸弹的二进制文件，可以直接运行，也可以用 gdb 调试；
- bomb.c：炸弹的 main 函数逻辑，主要用来让大家知道要对哪些函数进行反汇编；
- README：读一下就好了。

运行炸弹

通过命令 `./bomb` 运行炸弹：

```
sysu@debian:~/bomb1$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
_
```

这时程序会等待你输入第一关的通关字符串，如果你输入错误炸弹就会爆炸：

```
sysu@debian:~/bomb1$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
i don't know...

BOOM!!!
The bomb has blown up.
Your instructor has been notified.
sysu@debian:~/bomb1$ _
```

这个时候进入排行榜（http://ecop.jinhangdev.cn/bomblab-scoreboard/）查看，发现服务器上记录了你的炸弹爆炸了一次：

Bomb Lab Scoreboard

This page contains the latest information that we have received from your bomb. If your solution is marked **invalid**, this means your bomb reported a solution that didn't actually defuse your bomb.

Last updated: Sat Sep 14 16:50:40 2019 (updated every 5 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb1	Sat Sep 14 16:50	0	1	0	valid

Summary [phase:cnt] [1:0] [2:0] [3:0] [4:0] [5:0] [6:0] total defused = 0/1

这里需要说明的是，我们的排行榜仅 `Phases defused` 列作为同学们完成情况的监控，`Status` 列表示你的炸弹是否处于正常状态，其他列（爆炸次数和分数）均没有现实意义，无需关心。

调试炸弹

我们使用 gdb 工具来调试炸弹：

```
sysu@debian:~/bomb1$ gdb ./bomb
GNU gdb (Debian 8.2.1-2+b1) 8.2.1
Copyright (C) 2018 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
```

```
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bomb...done.
(gdb) _
```

当看到 `(gdb)` 时，表示 gdb 已经读取好了程序正等待你调试。常用的 gdb 命令如下：

- r (run)：启动程序运行（直到退出或遇到断点）
- c (continue)：继续程序运行（直到退出或遇到断点）；
- disas (disassemble)：用法为 `disas func_name`，对 func_name 函数进行反汇编；
- info r (info registers)：查看当前寄存器组状态（比如在程序 run 起来以后才能看到）；
- x：查看内存命令；通用格式为 `x/(nfu) addr`，`n` 是一个正整数，不写时默认为 1；`f` 是格式，若是字符串显示则写 `s`，不是字符串时通常可以不写；`u` 是单位字节数，默认为 4，可选的有 `b-单字节`，`h-二字节`，`w-四字节`，`g-八字节`。
  - 用法 1：`x/s addr`，查看 addr 地址开始的字符串（遇到 `\0` 结束）；
  - 用法 2：`x/8w addr`，查看 addr 处开始的连续的 8 个 4 字节数据；`x/4b addr`，查看 addr 处开始的连续 4 个字节……
- br (break)：设置断点，当程序每次运行到此处时均暂停返回 gdb 等待命令；
  - 用法 1：`br func_name`，在给定函数 func_name 的入口点处设置断点；
  - 用法 2：`br *addr`，在指定指令地址 `addr` 处设置断点，注意该 addr 必须是十六进制（0x12345678），且在 disas 命令的反汇编列表中显示的指令的首地址，否则将无法正常命中断点；
- expr (expression)：计算表达式，用法为 `expr 表达式`，如：`expr 1+1`，`expr $eax+5` 等。
- ni (nexti)：下一句汇编，即单步运行一行汇编语言。
- si (stepi)：下一步，即单步运行程序到下个 C 语言语句，由于我们在编译时没有将 C 语言代码加入进来，所以使用效果等同于 `ni`；
- q (quit)：退出 GDB，若被调试程序正在运行中，则会询问是否杀死进程，杀死进程则可以退出。

拆弹演示

首先我们得到整个炸弹的反汇编文件：

```
sysu@debian:~/bomb1$ objdump -d ./bomb > dump.txt
```

`objdump` 是标准反汇编工具，我们让反汇编的结果存入 `dump.txt` 这个文件中，如果觉得在虚拟机内不方便查看，可以传送到虚拟机外查看。

第一关

由 bomb.c 可知，每一关分别是一个函数，找到第一关的反汇编代码：

```
08048b5a <phase_1>:
 8048b5a:   55                      push   %ebp
```

```
 8048b5b:   89 e5                   mov    %esp,%ebp
 8048b5d:   83 ec 10                sub    $0x10,%esp
 8048b60:   68 84 a2 04 08          push   $0x804a284
 8048b65:   ff 75 08                pushl  0x8(%ebp)
 8048b68:   e8 30 05 00 00          call   804909d <strings_not_equal>
 8048b6d:   83 c4 10                add    $0x10,%esp
 8048b70:   85 c0                   test   %eax,%eax
 8048b72:   75 02                   jne    8048b76 <phase_1+0x1c>
 8048b74:   c9                      leave
 8048b75:   c3                      ret
 8048b76:   e8 56 07 00 00          call   80492d1 <explode_bomb>
 8048b7b:   eb f7                   jmp    8048b74 <phase_1+0x1a>
```

通过对反汇编代码逻辑的阅读，我们知道了函数吧两个参数传递给了 `strings_not_equal` 这个函数，这个函数顾名思义是比较两个字符串是否相同，有点类似于 `strcmp()` 函数，怀有疑惑的同学不妨去看看 `strings_not_equal` 的反汇编代码就好了。再来看爆炸的逻辑，0x8048b70 的指令将 `eax` 寄存器进行验证是否为零，不为零时跳到 `0x8048b76` 从而引发炸弹爆炸，那么只要只要不跳转就好了，于是就要 `eax` 为零，即 `strings_not_equal` 这个函数的返回值为 `0`，那么就让两个字符串相同就好了。这两个字符串分别是什么呢?

我们知道 `ebp+8` 这个位置是当前函数的第一个参数，由 bomb.c 知就是我们从屏幕输入的那个字符串，函数把第一个参数和一个地址 `0x804a284` 同时传入了比较函数，那么这个地址就非常可疑了，很可能就存储着用于比较的字符串。于是我们使用 gdb 的扫描内存功能扫描此处的字符串应该就能得到想要的结果了。运行 gdb，放置一个断点在 phase_1 的入口处，然后运行程序:

```
sysu@debian:~/bomb1$ gdb ./bomb
GNU gdb (Debian 8.2.1-2+b1) 8.2.1
Copyright (C) 2018 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".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bomb...done.
(gdb) br phase_1
Breakpoint 1 at 0x8048b60
(gdb) r
Starting program: /home/sysu/bomb1/bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
aaa

Breakpoint 1, 0x08048b60 in phase_1 ()
(gdb) _
```

由于输入字符串在进入 phase_1 之前，那么我们不得不随意输入一个字符串，使其能进入到 phase_1，现在用反汇编指令查看 phase_1 的函数体：

```
(gdb) disas
Dump of assembler code for function phase_1:
   0x08048b5a <+0>:     push   %ebp
   0x08048b5b <+1>:     mov    %esp,%ebp
   0x08048b5d <+3>:     sub    $0x10,%esp
=> 0x08048b60 <+6>:     push   $0x804a284
   0x08048b65 <+11>:    pushl  0x8(%ebp)
   0x08048b68 <+14>:    call   0x804909d <strings_not_equal>
   0x08048b6d <+19>:    add    $0x10,%esp
   0x08048b70 <+22>:    test   %eax,%eax
   0x08048b72 <+24>:    jne    0x8048b76 <phase_1+28>
   0x08048b74 <+26>:    leave
   0x08048b75 <+27>:    ret
   0x08048b76 <+28>:    call   0x80492d1 <explode_bomb>
   0x08048b7b <+33>:    jmp    0x8048b74 <phase_1+26>
End of assembler dump.
(gdb)
```

其中，`=>` 所指向的语句就是当前程序暂停的位置。我们对上面讨论的地址进行内存扫描，得到：

```
(gdb) x/s 0x804a284
0x804a284:      "I turned the moon into something I call a Death Star."
(gdb)
```

好像得到了答案，我们记下答案，并重新运行炸弹：

```
sysu@debian:~/bomb1$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I turned the moon into something I call a Death Star.
Phase 1 defused. How about the next one?
```

发现我们的第一关已经成功拆解，刷新一下排行榜将看到服务器已经顺利记录：

Bomb Lab Scoreboard

This page contains the latest information that we have received from your bomb. If your solution is marked **invalid**, this means your bomb reported a solution that didn't actually defuse your bomb.

Last updated: Sat Sep 14 18:29:20 2019 (updated every 5 secs)

#	Bomb number	Submission date	Phases defused	Explosions	Score	Status
1	bomb1	Sat Sep 14 18:28	1	1	10	valid

Summary [phase:cnt] [1:1] [2:0] [3:0] [4:0] [5:0] [6:0] total defused = 0/1

由于我们现在不知道第二关的答案，需要退出程序进入调试模式，于是我们使用 `^C`（即 `Ctrl-C`） 键退出程序：

```
sysu@debian:~/bomb1$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I turned the moon into something I call a Death Star.
Phase 1 defused. How about the next one?
^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
sysu@debian:~/bomb1$
```

程序会输出一些调皮的话，并返回操作系统。

简易操作

由于第一关我们已经拆解，每次都手工输入字符串会显得非常麻烦，于是我们将已经拆解的答案存入一个文档 `solution.txt` 中，该文档结构如下：

```
answer for phase_1
answer for phase_2
answer for phase_3
answer for phase_4
answer for phase_5
answer for phase_6
answer for secret_phase
```

即每关一行字符串，当前做完几关就写几行，程序发现文件不够读时会自动转为从控制台读入。运行程序时，则使用命令行

```
sysu@debian:~/bomb1$ ./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?
_
```

即可。

解题提示

- 从第二关开始，每关的字符串形式要根据 `scanf` 的格式化字符串确定，我们知道形如 `"%d %d"` 这样的格式化字符串本身也是一个字符串，你需要用类似第一关的方法挖掘出来，知道本关要输入的内容是什么，然后看这些格式化读入的数据分别存入了哪个变量（寄存器或栈中内存地址），并以此对程序的发展进行追踪，最后得到本关的答案。
- 鼓励根据反汇编代码反推高级语言伪代码，加深对汇编语言和高级语言的关系的理解，后面的关卡可能涉及到递归、链表等 C 语言现象，写出伪代码将帮助你更快更准确地解题！
- 学好汇编语言，是程序猿/媛的基本功~！

--by Jin Hang, 2019.9.14