Bomblab Preview

在这个 Lab 中，你会尝试拆除一个“二进制炸弹”—— 实际上是一个包含多个关卡的问答程序。每个问题都要求你输入一个字符串，答对所有问题就能得到满分。然而，你能拿到的只有一个可执行文件，意味着你需要通过**反汇编**的方式解析程序逻辑，逆向工程找出答案。你将学会熟练使用`gdb`，并对C语言映射的汇编语言有一个比较全面的理解

。以下是**非常重要**的tips：

每个人的Bomb是**随机生成**的，意味着你不能挪用别人的答案
请在 class machine 上完成拆弹，否则服务器不能正确记录分数！
在任何阶段，解答错误都会导致本lab得分**永久扣除**！不用太过担心，接下来会介绍拆弹的“安全措施”；但必要张感还是要有的，毕竟你的误操作可能无法挽回。

`bomb.c`

这只是Bomb的大体框架，你可以看到六个问题分别所在的函数名字，方便你的`gdb`操作；其余的信息对拆弹并无帮助。

`bomb`

你要处理的二进制炸弹。为了你的(分数)安全着想，请不要一上来就直接运行它。下面是利用`gdb`的**安全措施**：

首先用`objdump`反汇编一下bomb：

```
objdump -d bomb > bomb.s     # 方便起见，保存到文件，之后可以随时查看
cat bomb.s
```

在输出的一长串汇编代码中(不要担心，你不需要看懂全部内容)，找到这样一个引人注目的函数名：

```
0000000000002108 <explode_bomb>:
    2108:       f3 0f 1e fa             endbr64
    210c:       55                      push   %rbp
    210d:       53                      push   %rbx
    210e:       48 83 ec 68             sub    $0x68,%rsp
    2112:       64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
    2119:       00 00
    211b:       48 89 44 24 58          mov    %rax,0x58(%rsp)
```

顾名思义，这是你引爆炸弹时调用的函数；它会连接服务器，给你的lab扣上分数。为了让炸弹安全化，你只需要**总是用`gdb`运行炸弹，并事先在这里打上断点**：

```
gdb -q bomb     # 用 gdb 加载bomb;  -q表示quiet，否则gdb会输出一长串没用的信息
```

此时你应该看到

```
Reading symbols from bomb...done.
(gdb)
```

要插入断点，只需要输入

```
(gdb) b explode_bomb
```

b是breakpoint的缩写；你应该看到

```
(gdb) b explode_bomb
Breakpoint 1 at 0x2108
(gdb)
```

具体地址可能引人而异；不过重点是，你现在已经不需要担心炸弹引爆了！举个例子，用`run`运行炸弹，给第一个问题 随便输句答案("ICS is hard"):

```
(gdb) run
Starting program: /home/CA11/bomb

Welcome to Dr. Evil's little bomb. You have 6 phases with
which to blow yourself up. Have a nice day ! Mua ha ha ha !
ICS is hard

Breakpoint 1, 0x000055c9c1716108 in explode_bomb ()
(gdb)
```

可以看到答案错误，已经触发`explode_bomb`处的断点。此时，你只需要从容地输入`kill`并选择yes，就可以安全地再次尝试了！

```
Breakpoint 1, 0x000055c9c1716108 in explode_bomb ()
(gdb) kill
Kill the program being debugged? (y or n) y
(gdb)
```

注意：你每次重新用 `gdb` 打开bomb，都需要重新设置断点！！！

你可能会用到的工具

- `objdump -d bomb` 可以一次性反汇编整个程序；用重定向可以把输出保存到文件，方便你检索。
- `objdump -s -j xxx bomb` 可以展示程序特定的section；你可能需要用到的是 .data和.rodata；前者保存全局变量值，后者保存字符串、 switch跳转表等常量值。
- `gdb` 的简单用法参见课上slides；详细教程参见Writeup。