

**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 ： 本硕博202001**

**学 号 ： U202115666**

**姓 名 ： 刘文博**

**指导教师 ： 许向阳**

**2022 年 10 月 1 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6.1** 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。

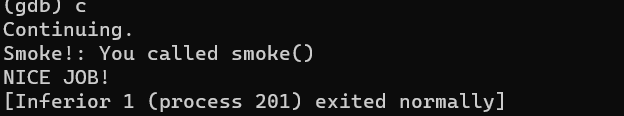
构造攻击字符串，让函数 getbuf将cookie值返回给 test函数，而不是返回值 1 。还原被破坏的栈帧状态，将正确的返回地址压入栈中，并且执行 ret 指令，从而返回到 test函数。

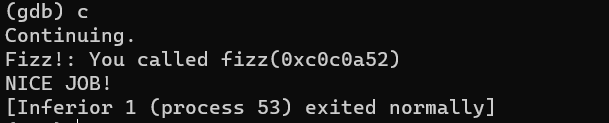
**5、第4级 kaboom**

**一个函数的栈帧的地址通常并不是固定的，随程序运行实例的不同而不同，即每次运行有一个随机的、不固定的值。在此种条件下，要求getbuf（getbufn）函数返回cookie的值，而不是返回值1，并且能正确回到调用函数处继续执行。**

**三、实验记录及问题回答**

**（1）实验任务 6.1 的实验记录**





**（2）缓冲区溢出攻击中字符串产生的方法描述**

阅读bufbomb.c文件发现根据最后一位学号为6，w=-9

所以根据自己的学号最后一位进行条件编译

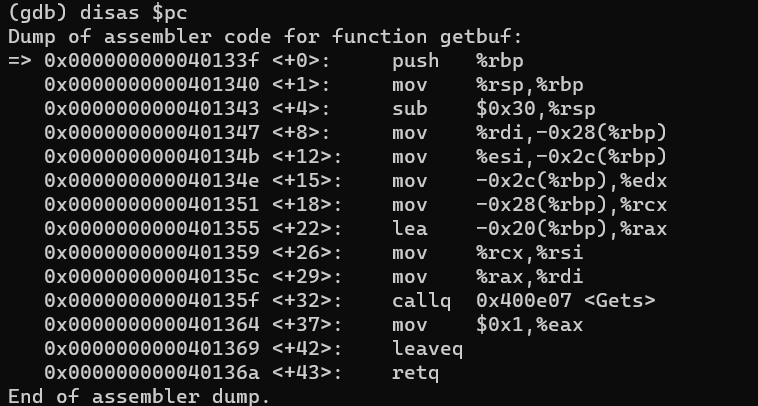
gcc -g bufbomb.c buf.c support.c -o bufbomb -D U6

之后设置参数 set args U202115666 smoke\_hex.txt 0

同时发现当level为0~3级时，主调函数test从文件或输入中读取字符串保存到byte\_buffer，接着会调用val = getbuf(byte\_buffer, byte\_buffer\_size);如果没有阻止getbuf回到test，val将等于cookie，炸弹爆炸。

之后进行反汇编

得到getbuf的反汇编代码



由于

int getbuf(char \*src, int len)

{

char buf[NORMAL\_BUFFER\_SIZE];

Gets(buf,src,len);

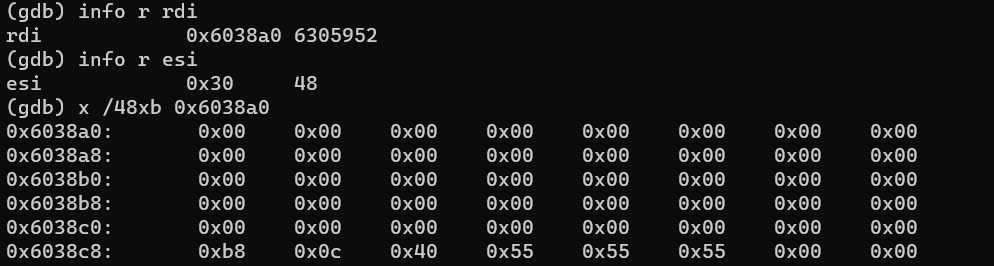
return 1;

}

0x0000000000401347 <+8>: mov %rdi,-0x28(%rbp)

0x000000000040134b <+12>: mov %esi,-0x2c(%rbp)

故rbp-0x2c处保存了len，rbp-0x28处保存了\*src

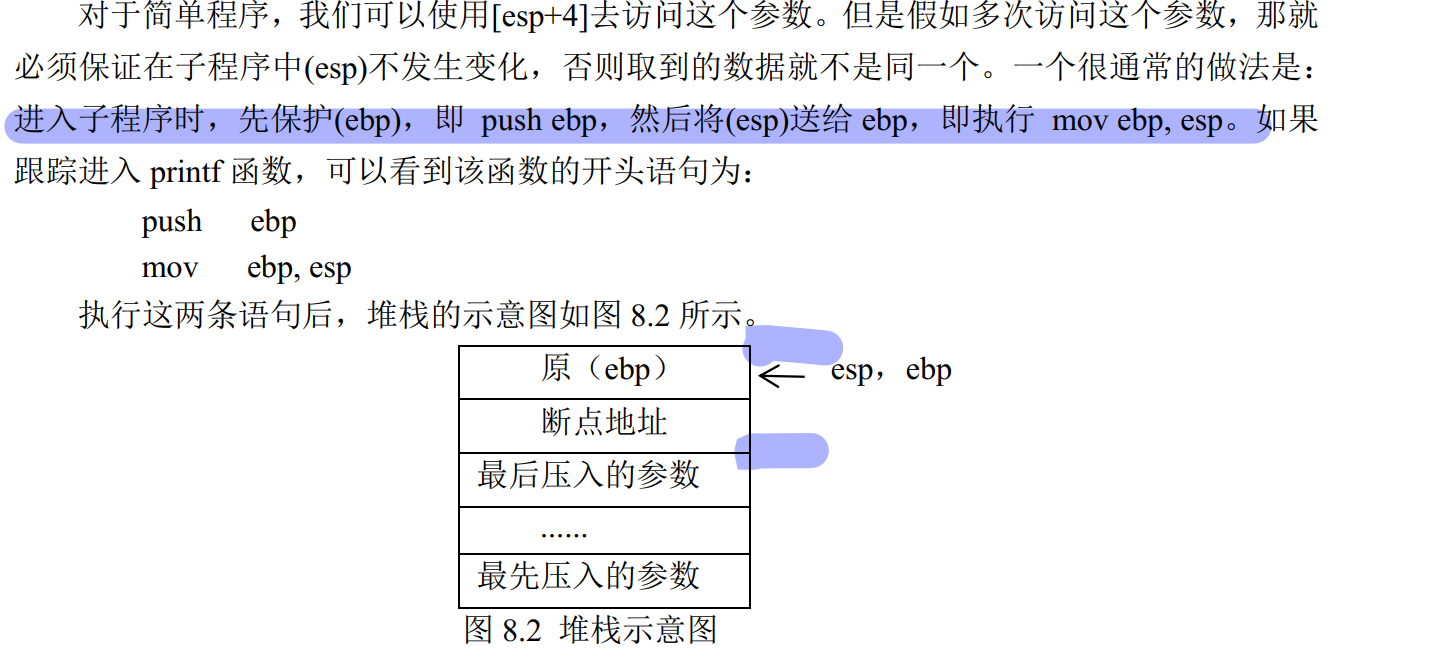


反汇编结果也验证了我们的猜想。

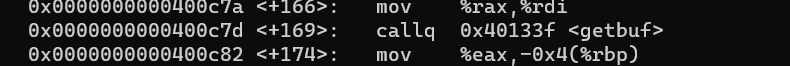
由#define NORMAL\_BUFFER\_SIZE 32 故getbuf的堆栈段有为buf预留了32个字节，有4个字节保存getbuf函数的返回地址，还要将\*src和len两个参数压栈。

根据Gets函数的三个参数buf，src，len，以及相关寄存器的传值，可以推测出调用Gets函数之前buf的首地址就是rbp-0x20，那么从rbp-0x20到rbp一共32位，保存buf。

根据汇编语言所学，进入getbuf函数会将rip的值送给rsp，之后会用rbp来保护rsp。其在堆栈段的结构如图所示



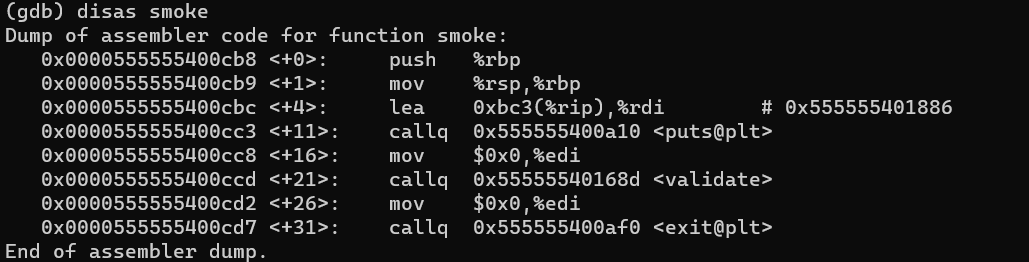
所以rbp+8即为保存rip的位置。



而调试的结果也验证了我们的猜想



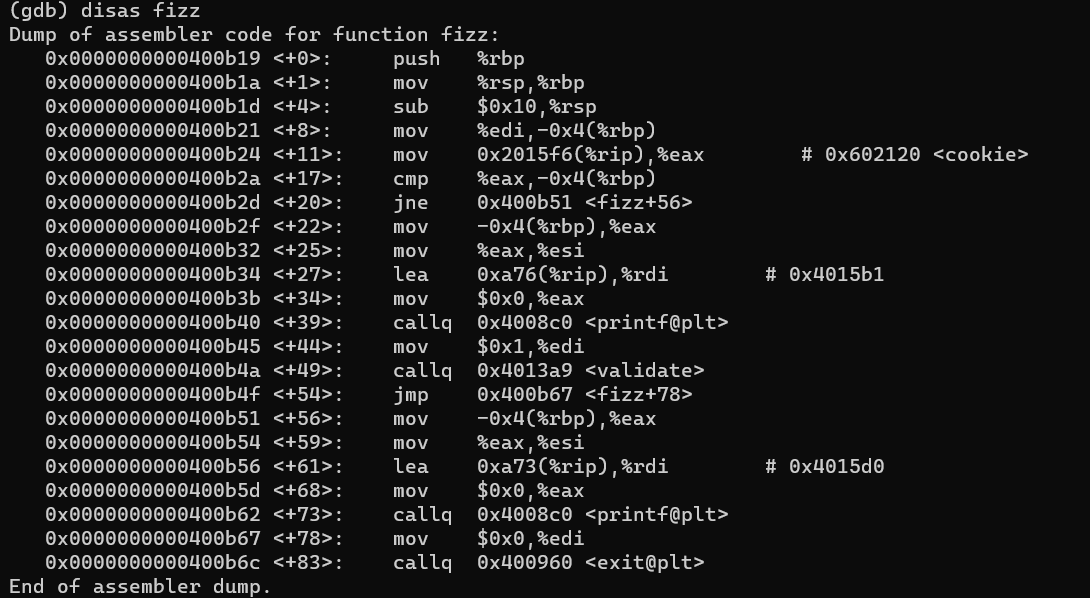
所以只需用溢出攻击将rbp+8之后的内容修改为smoke的入口地址

反汇编smoke得到

所以修改smoke\_hex.txt的内容为40个00（buf占32字节，rbp占8字节）加上smoke的8字节入口地址即可。

接着进行fizz，和smoke类似

反汇编fizz得到



根据fizz函数的参数val，和

0x0000000000400b21 <+8>: mov %edi,-0x4(%rbp)

0x0000000000400b24 <+11>: mov 0x2015f6(%rip),%eax #0x602120 <cookie>

容易知道edi中保存了val的值会与0x602120处的cookie作比较。然而这里我们不能使用edi的值，因为它很可能会出错。

这里用溢出攻击将返回地址修改到下面的语句

0x0000000000400b24 <+11>: mov 0x2015f6(%rip),%eax #0x602120 <cookie> ，这样紧接着会执行

0x0000000000400b2a <+17>: cmp %eax,-0x4(%rbp)，

而rbp-0x4的值我们可以修改为0x602120，这样就可以满足判断条件。

所以修改smoke\_hex.txt的内容为32个00，再跟上0x602120+4的8字节值（buf占32字节，rbp占8字节），再加上fizz中相应语句的8字节地址即可。

bang和kabang的实现从略。

**四、体会**

通过对缓冲区炸弹的破解，我深刻领会到了不检查输入长度可能对堆栈产生的巨大影响，同时熟悉了子程序的结构，和rip与堆栈中rsp、rbp的联系，对条件编译和传参函数的调试有了更深入的掌握。