

**课 程 实 验 报 告**

**课程名称： 计算机系统基础实验**

**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 ： CS2205**

**学 号 ： U202215510**

**姓 名 ： 徐新飏**

**指导教师 ： 李专**

**2024 年 4 月 26 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**

**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）实验任务的实验记录**

生成程序，执行指令，用objdump -d bufbomb > bufbomb.s生成存储整个程序反汇编代码的文本文件。

阶段0：smoke

构造攻击字符串作为目标程序输入，造成缓冲区溢出，使getbuf()返回时 不返回到test函数，而是转向执行smoke。

getbuf函数如图所示

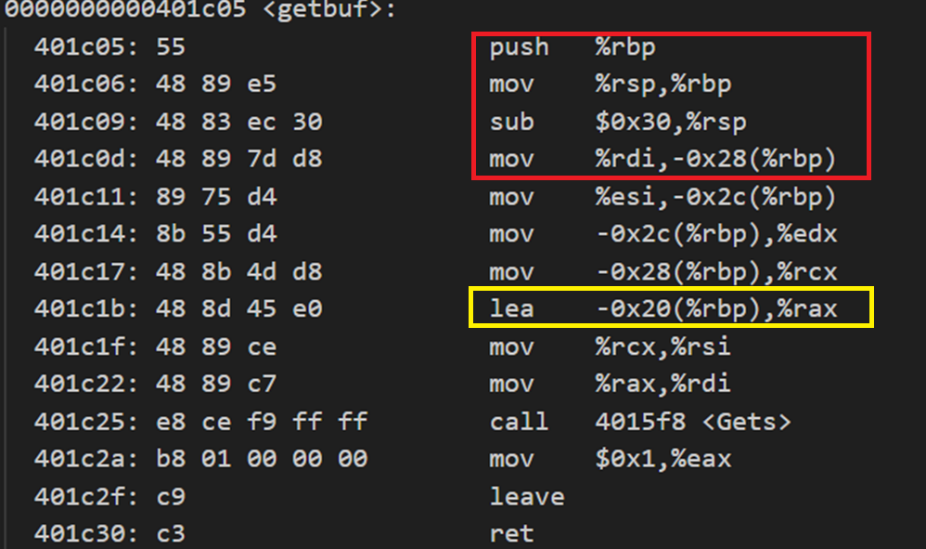


图1.1

注意到红色方框的部分为开拓0x30的空间，故rsp栈帧空间为0x30+0x8=0x38个字节空间。黄色方框部分，lea-0x20(%rbp)，%rax是创建的局部变量buf所在空间，空间大小为0x20个字节。

要使其缓冲区溢出，且要覆盖此栈帧中原rbp值以及断点返回值的信息，则要构造至少0x20+0x8+0x8=0x30字节长度的字符串。攻击字符串后8个字节要覆盖返回地址值，故设置其为smoke首地址0x4013e8，构造溢出攻击字符串。

根据小端存储的性质，其应当最后返回为e8 13 40 00 00 00 00 00。

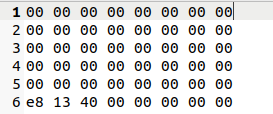


图1.2

将构造好的攻击字符串输入到txt文档中，运行程序发现通过了第一关。

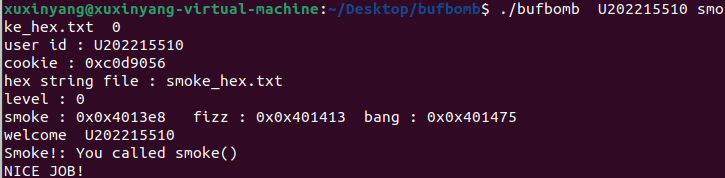


图1.3

阶段1：fizz

观察到fizz函数的首地址为0x401413，与smoke的差别是，fizz函数比较了val和cookie的值。

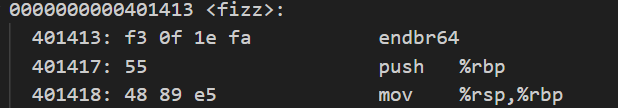


图1.4

查看当前函数栈帧构造方式，可以跟踪看出当前cookie的值位于%rip+0x2d20处，故除用输入攻击字符串覆盖返回地址值外，例如阶段1中的工作，还需要构建进入该函数前的参数。因为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。

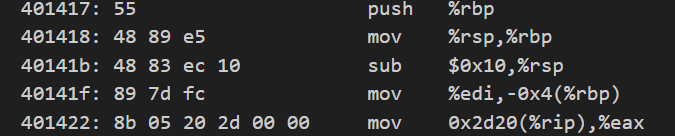


图1.5

通过gdb观察反汇编代码可知，fizz函数的参数val通过edi传参并存入-0x4(%rbp)中，在bufbomb.c文件中设置断点，获取cookie地址值为0x404148，将cookie地址加4即得到要修改rbp的值为0x40414c。

将要修改的rbp的值整合进入上一关的攻击字符串，接入将 rip赋值给if语句地址0x401422，整合攻击串如图。

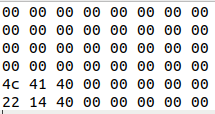


图1.6

通过测试。

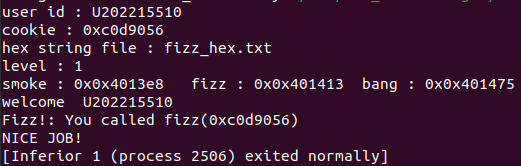


图1.7

阶段2：bang

构造攻击字符串，使目标程序调用bang函数，要将函数中全局变量 global\_value 篡改为 cookie 值，使相应判断成功，需要在缓冲区中注入恶意代码篡改全局变量。

构造字符串覆盖getbuf栈帧缓冲区，使缓冲区溢出，先返回到恶意代码位置，即攻击字符串位置修改全局变量，再跳转到bang函数中，恶意代码的机器码通过gcc指令获得，局部变量首地址通过gdb指令获得。生成的机器码如图。

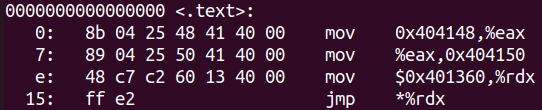


图1.8

gdb和直接运行模式下输出的buf首地址，在直接运行模式下，是通过修改buf.c源码的方式，使其打印出buf的首地址。





图1.9

将反汇编的机器码存入buf中，缓冲区溢出后首先转到buf的首地址，然后执行buf中的机器码，执行后跳转到bang的比较语句，构造攻击串如下。

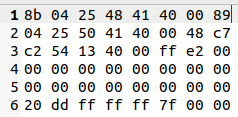


图1.10

攻击串测试通过

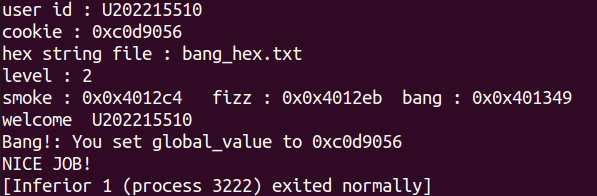


图1.11

阶段3：boom

首先查找test函数的中调用getbuf函数指令的下一条指令地址，为0x401469, 实现攻击之后，要返回到getbuf的下一条指令。要求让函数 getbuf将cookie值返回给test函数，而不是返回值1。



图1.12

构造汇编程序段实现攻击，将cookie的值传给eax，然后将程序跳转到call getbuf的下一条语句。

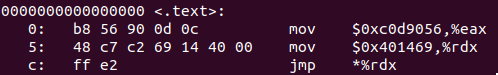


图1.13

由于要栈帧中原rbp不被破坏，则需要获取原rbp值，通过gdb查看原rbp值。



图1.14

构造gdb模式下攻击字符串如下，随后进行攻击测试。

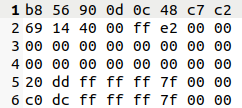


图1.15

在gdb模式下利用刚才获取rbp值，测试结果如图。

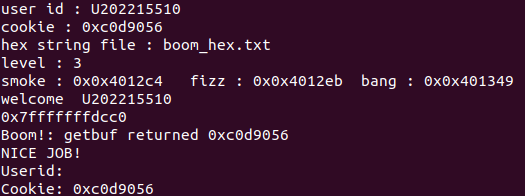
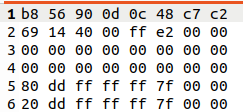


图1.16

在直接运行模式下，注意到gdb模式下原rbp值与buf首地址值相差60，那么由直接运行模式下的buf首地址+0x60就可以推出直接运行模式下rbp的值，构造攻击字符串后运行通过。



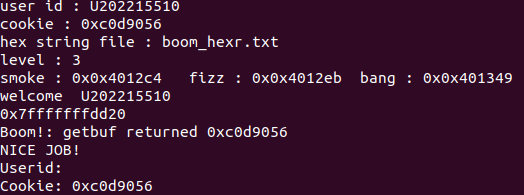


图1.17

**（2）缓冲区溢出攻击中字符串产生的方法描述**

阶段0：smoke

构造攻击字符串作为目标程序输入，造成缓冲区溢出，使getbuf()返回时 不返回到test函数，而是转向执行smoke,只需要将原先栈中存储的rip换为函数smoke的地址。

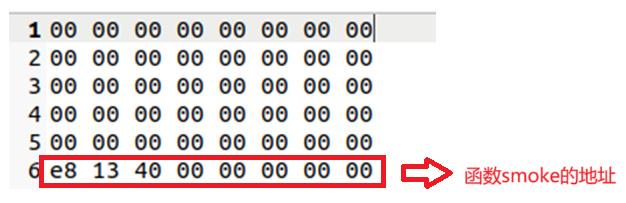


图2.1

阶段1：fizz

除用输入攻击字符串覆盖返回地址值外，例如阶段1中的工作，还需要构建进入该函数前的参数。因为fizz函数中比较了参数val与全局变量cookie的值，只有两者相同才能达到目标。

获取cookie地址值为0x404148，将cookie地址加4即得到要修改rbp的值为0x40414c，将要修改的rbp的值整合进入上一关的攻击字符串。

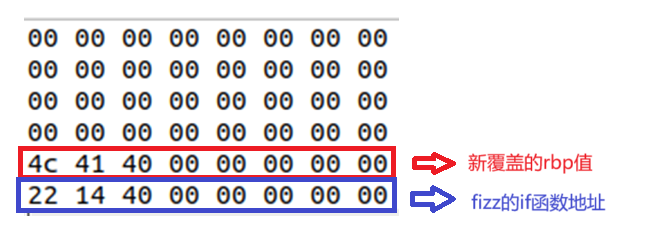


图2.2

阶段2：bang

构造字符串覆盖getbuf栈帧缓冲区，使缓冲区溢出，利用溢出后的rip先返回到恶意代码位置，即攻击字符串位置修改全局变量，再跳转到bang函数中。

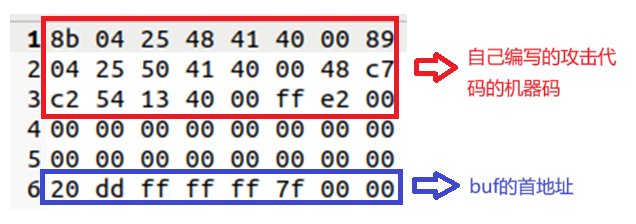


图2.3

阶段3：boom

实现攻击之后，要返回到getbuf的下一条指令，同时要求让函数 getbuf将cookie值返回给test函数，为实现无感攻击，也要保持初始的rbp值正确。

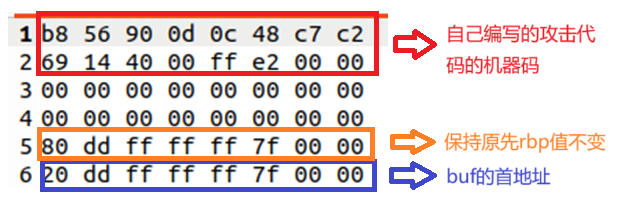


图2.4

**四、体会**

完成缓冲区炸弹实验bufbomb使我意识到这一实践在计算机安全领域的应用，加深了我对于汇编语言和计算机系统基础的理解和技能的提升。

通过分析缓冲区炸弹程序的源代码和反汇编结果，我更深入地理解了程序是如何在底层机器级别上执行的，包括函数调用、栈帧布局、指令执行顺序等各种方面的细节。在实验中，我也更加熟悉了gdb的使用，熟练掌握了诸如如何设置断点、单步执行、查看内存和寄存器状态等技巧。

更重要的是我加强对缓冲区溢出的理解，了解到缓冲区溢出是一种常见的安全漏洞，能够导致严重的安全问题。解决问题的过程中需要综合运用多种技能，锻炼的问题解决能力和逻辑思维能力，使我更有信心地面对今后的技术挑战，这是一次非常有价值的学习和成长经历。