

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**实验名称：机器级语言理解(缓冲区溢出攻击)**

**院 系 ：计算机科学与技术**

**专业班级 ： CS2305班**

**学 号 ： U202315677**

**姓 名 ： 岳皓**

**指导教师 ： 班鹏新**

**2025 年 4 月 10 日**

# 实验四 机器级语言理解(缓冲区溢出攻击)

**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)等。

**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。

**三、实验记录及问题回答**

**（1）实验任务的实验记录**

先用指令gcc -g -o bomb -D U7 bufbomb.c support.c buf.c构建可执行文件bomb，再用指令objdump -d bufbomb > bufbomb.asm将bufbomb的反汇编代码放在生成的bufbomb.asm文件中，便于查看。

可以输入下列指令去掉一些栈保护和一些检查代码的生成：

gcc -g -fno-stack-protector bufbomb.c buf.c support.c -o bufbomb

gcc -g -fno-stack-protector -no-pie bufbomb.c buf.c support.c -o bufbomb -DU7

gcc -g bufbomb.c buf.c support.c -o bufbomb -D U7

gcc -g -D U7 -fno-stack-protector -no-pie -fcf-protection=none bufbomb.c buf.c support.c -o bufbomb

gcc -g -D U7 -fno-stack-protector -no-pie -fcf-protection=none -z execstack bufbomb.c buf.c support.c -o bufbomb

gcc -g -D U7 -fno-stack-protector -no-pie -fcf-protection=none bufbomb\_new.c buf.c support.c -o bufbomb\_new

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。实验要求构造攻击字符串，使得getbuf()返回时并非正常返回，而是转向执行smoke函数。

首先，在bufbomb.asm文件中查找并观察getbuf函数的反汇编代码，如图1所示。

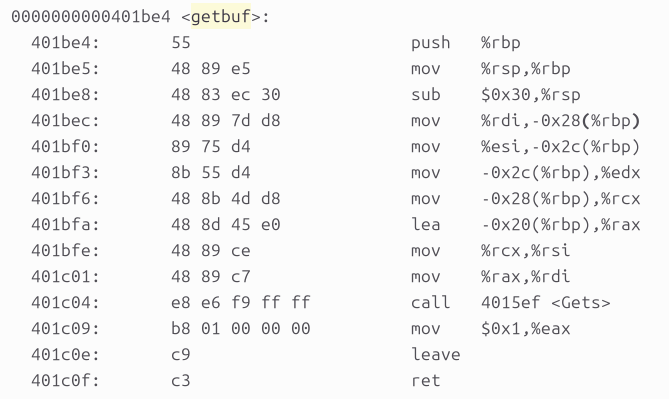
****

图 1 getbuf函数反汇编代码

从图中可知，buf缓冲区的大小为0x0，即32个字节。再查找并观察smoke函数的反汇编代码，如图2所示，由图可知，smoke的首地址为00000000004012bb。

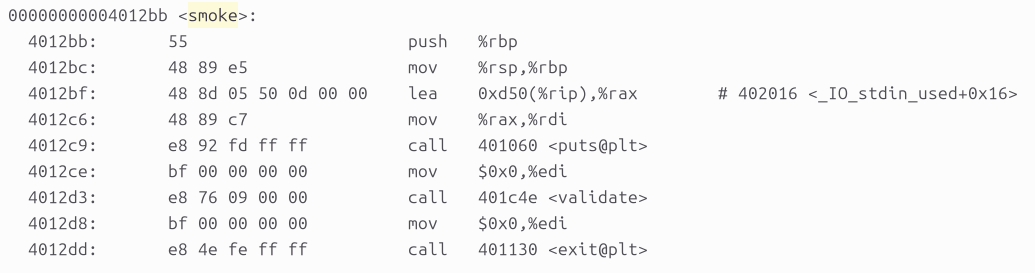
****

图 2 smoke函数反汇编代码

按照要求，需要使攻击字符串的大小超出数组buf的大小，由此溢出来覆盖原先rbp的值以及rbp上方的返回地址，故而攻击字符串的大小应为0x30+8+8 = 64个字节。并且最后的8个字节为smoke函数的入口地址，而其余字节我们可以全使用0来填充，如图3，将构建的攻击字符串放在创建的smoke\_hex.txt中。

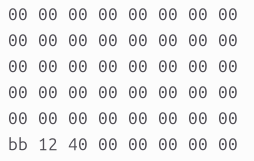
****

图 3 攻击字符串设计

然后输入指令/bufbomb U202315677 smoke\_hex.txt 0来运行，结果如图4。

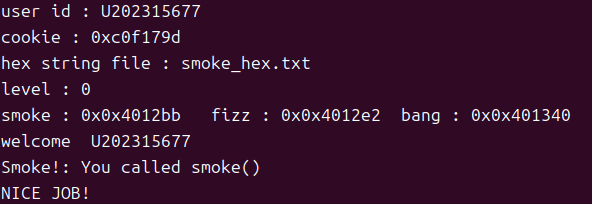
****

图 4 通关截图

**2、第1级 fizz**

构造攻击字符串，使得getbuf()返回时并非正常返回，而是转向执行fizz函数，同时还需要比较参数val与全局变量cookie的值，需要两者相同。

首先，观察fizz函数的反汇编源代码，如图5所示。

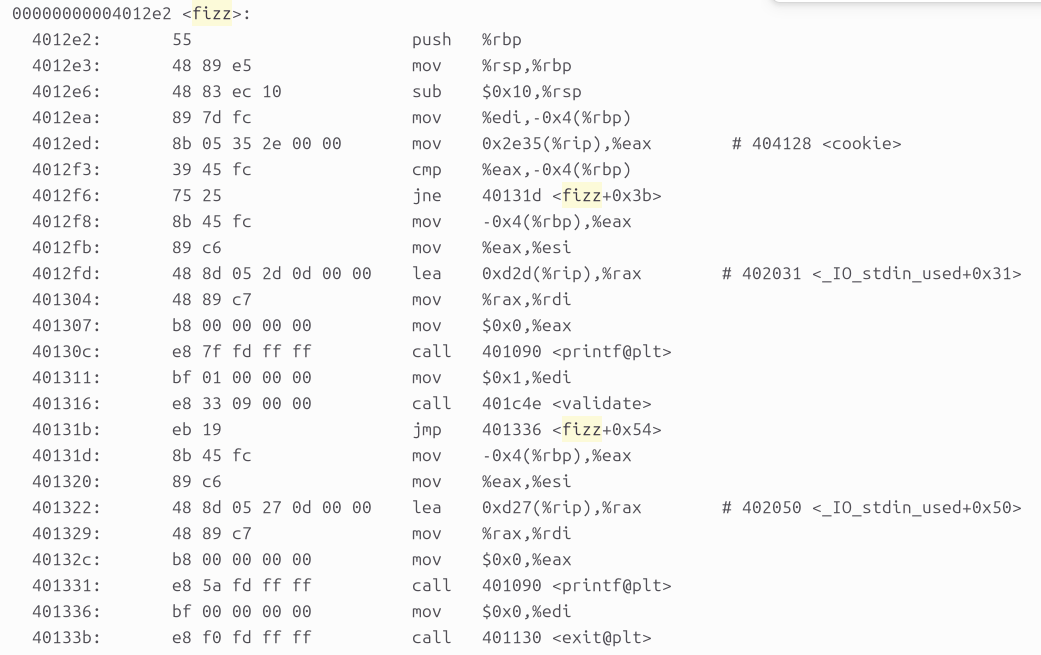
****

图 5 fizz函数反汇编代码

从图5中，我们可以知道fizz函数的入口地址为0x00000000004012e2。故而转向fizz函数的方式可以同”smoke”关卡一般修改返回地址即可。但是在这里并不能直接跳转到fizz的首地址处。

对于64位程序，使用的是寄存器EDI来传递int型的参数val。直接修改EDI是很困难的。一种巧妙的办法是，不要跳到 fizz函数的起始地址，而是跳到401341处，如果只是简单绕开if语句的检查，之后的打印语句打印val的值会发现它与cookie并不一致。在fizz函数中只要%rbp-0x4与cookie对应同一个单元，则if的条件就会成立，但是这里的RBP的值并不是我们getbuf函数中RBP的值。这里需要正确设置RBP的值，使得%rbp-0x4与cookie对应同一个内存单元。我们需要将getbuf函数栈帧中的RBP旧值进行修改，并且修改成cookie的地址+0x4，这样可以使得在fizz函数中%rbp-0x4和cookie对应一个内存单元。

根据栈帧结构，我们需要从第48个字节写入cookie的地址+0x4，地址占8个字节，cookie地址是0x404128,cookie地址+0x4即0x40412c,从4012ed处写入我们要跳转的fizz函数位置，地址为0x00000000004012ed。

所以构建如图6的攻击字符串并保存在fizz\_hex.txt中。

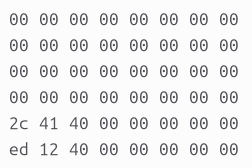
****

图 6 攻击字符串设计

在终端中输入指令./bufbomb U202315677 fizz\_hex.txt 1来运行，结果如下图。

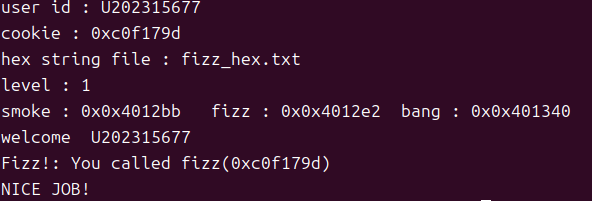
****

图 7 通关截图

1. **缓冲区溢出攻击中字符串产生的方法描述**

**第0级 smoke：**

对我们程序起作用的就是getbuf函数的堆栈，所以我们只需要画出调用Gets函数前后的堆栈情况即可，这里不以具体的地址表示，用刚刚进入getbuf后新的rbp作为标准。

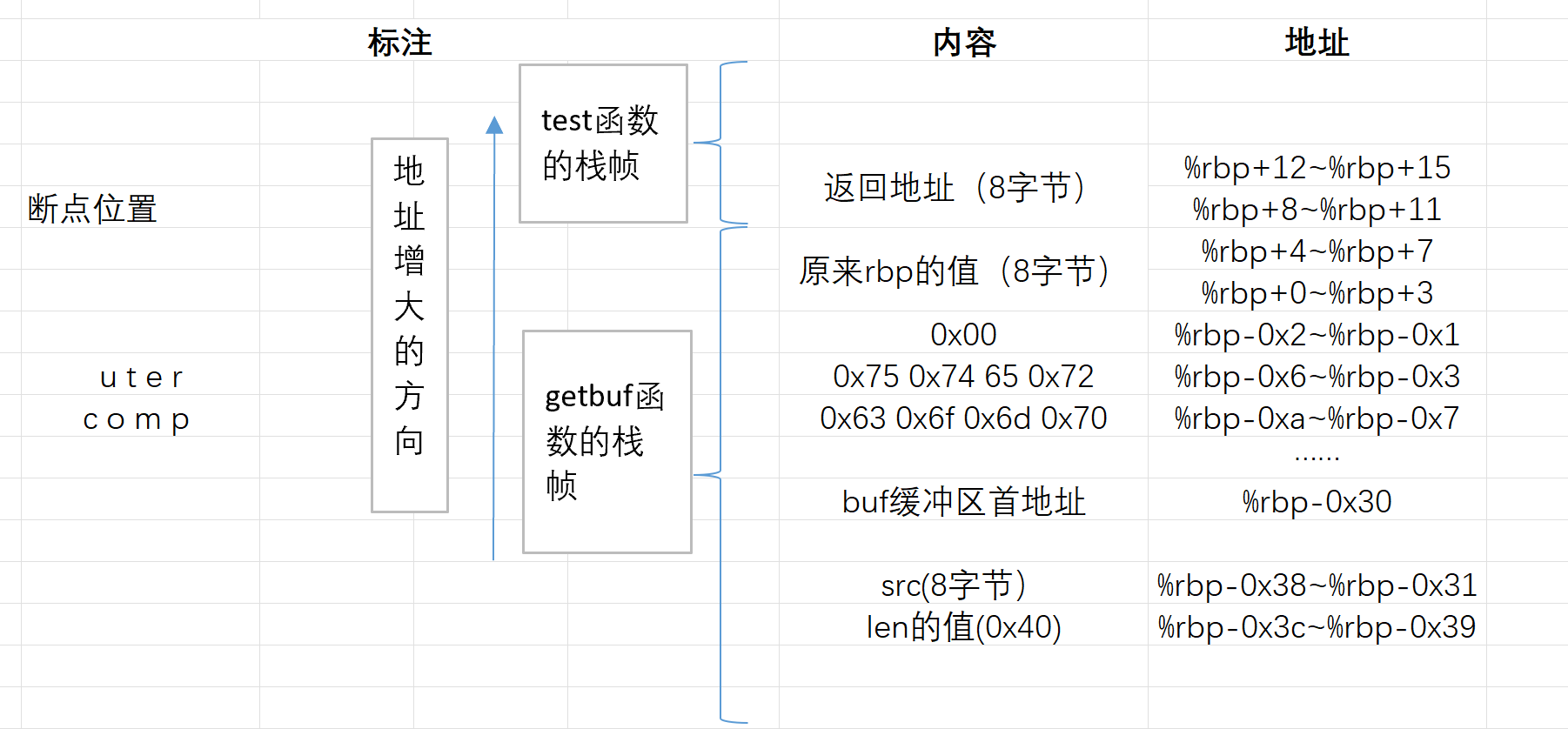
****

图 8 未调用Gets前的堆栈



图 9 调用Gets后的堆栈

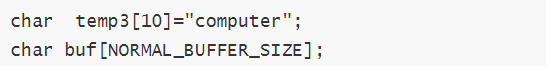


图 10 getbuf中的局部变量

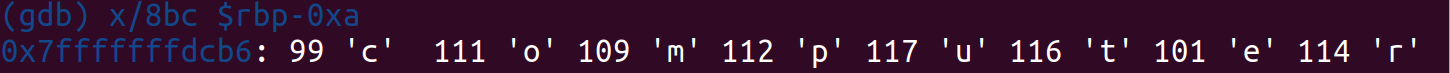
****

图 11 局部变量temp3

简单分析一下，堆栈中有temp3和buf两个局部变量，temp3的首地址是%rbp-0xa,buf的首地址是%rbp-0x30。我们还需要注意断点在%rbp+0x8的位置，rbp的存储位置在%rbp。

**第1级 fizz：**

这一关的堆栈情况和上一关基本一致，区别在于将栈帧中RBP的旧值修改为cookie地址+0x4，这个存放的位置由计算得来，具体可以参考实验内容部分。

1. **体会**

在进行缓冲区溢出实验的过程中，我获得了宝贵的经验和深刻的体会，以下是我的总结：

首先，我深刻认识到了安全漏洞的严重性。缓冲区溢出漏洞对计算机系统安全构成了巨大威胁，可以导致执行恶意代码、获取敏感信息甚至控制系统。通过实验，我们仅仅利用了一个数组的输入溢出问题就导致了各种函数之间的跳转以及常量的被修改，这让我深刻意识到了这种漏洞的危害性，加强了我对这方面知识的学习和认识。

其次，我学习了缓冲区溢出的原理和技术，包括栈溢出、堆溢出等常见类型。深入了解攻击者利用溢出漏洞实施攻击的方法和手段，实践实现了一些常见的攻击方式，如smoke和fizz。了解到这些攻击方式背后的原理和实现机制，有助于我将来面对攻击时能够更好地找到防御的方向，例如利用栈溢出来覆盖返回地址从而实现程序流程控制。

我还初步掌握了调试和分析工具的使用方法，如GDB调试器、objdump反汇编工具等。这些工具帮助我深入理解程序的执行过程，分析攻击手段，并且使我能够成功自己编写一些简单的缓冲区溢出攻击代码。例如，通过GDB调试器，我可以观察程序栈的变化情况，从而找到溢出点，并利用此信息进行攻击代码的编写。

通过实验，我不仅提升了自己的安全意识和技术能力，还加强了对计算机系统安全的认识和理解。加强了团队合作和沟通，与同学们共同合作、相互交流学习，共同解决问题。这些经验对我未来的学习和工作产生了积极的影响，使我能够更好地应对安全挑战和保护系统安全。