**《软件安全》实验报告**

姓名：胡博浩 学号：2212998 班级：信息安全

**一、实验名称：**

Shellcode编写及编码

**二、实验要求：**

复现第五章实验三，并将产生的编码后的shellcode在示例5-4中进行验证，阐述shellcode编码的原理、shellcode提取的思想。

**三、实验过程：**

1.第五章实验三复现：

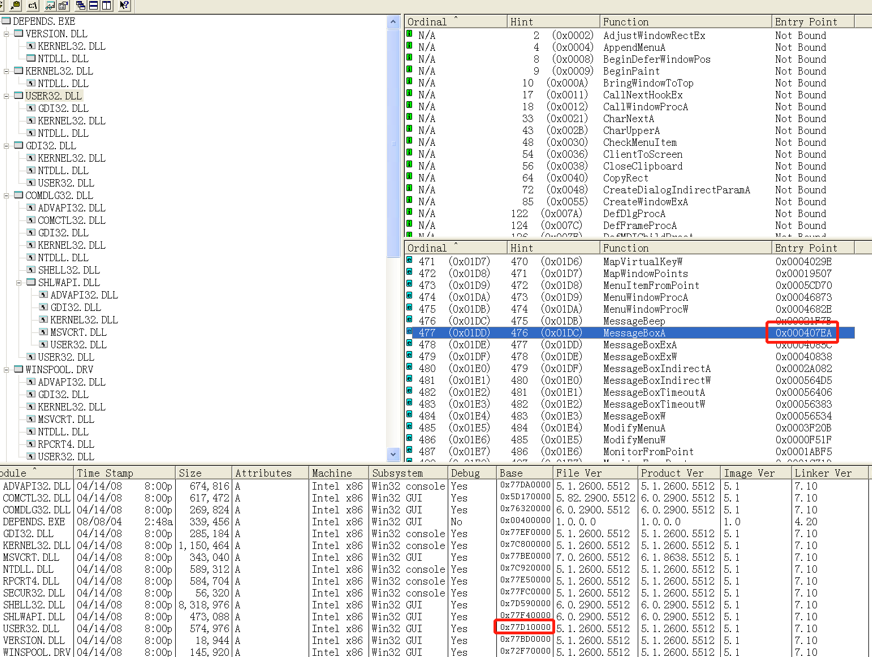
实验要求将一个能够弹出“Hello world”的shellcode进行编码，再把编码后的shellcode和一个解码的shellcode合并成最终的shellcode，从而进行利用。

那么首先要编写出一个弹出显示“Hello world”消息框的shellcode。让程序弹出一个消息框，只需要调用user32.dll中的MessageBoxA函数。

因此第一步是获得MessageBoxA函数入口地址。有两种方法可以获取到MessageBoxA的函数入口地址：

（1）通过工具获取user32.dll在系统中加载的基地址和MessageBoxA函数在库中的偏移来确定函数最终的地址。

效果如下图所示，得到的MessageBoxA方法在内存中的地址为0x77D507EA。

****

（2）使用代码来获取相关函数地址：即通过程序来动态获取，代码如下。

#include <windows.h>

#include <stdio.h>

int main(){

HINSTANCE LibHandle;

FARPROC ProcAdd;

LibHandle = LoadLibrary("user32");

//获取user32.dll的地址

printf("user32 = 0x%x \n", LibHandle);

//获取MessageBoxA的地址

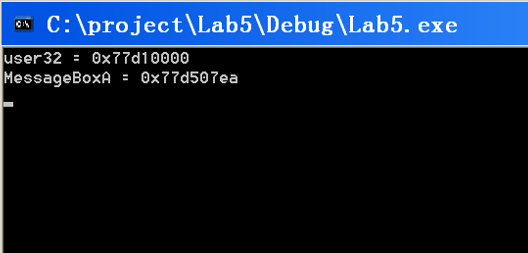
ProcAdd=(FARPROC)GetProcAddress(LibHandle,"MessageBoxA");

printf("MessageBoxA = 0x%x \n", ProcAdd);

getchar();

return 0;}

运行后结果如下。



得到了MessageBoxA的入口地址之后，我们就可以将函数所需要的4个参数依次入栈，然后通过call指令直接调用函数。

为了能够在弹出的消息窗口中展示出“Hello World”，我们需要将“Hello World”的字符串压入到栈中，入栈需要倒序，考虑到大端编码，存储顺序也要反过来。在最后补上空格凑足字节，并且首先压入了一个0，保证字符串有串尾符\0。

由于shellcode 中最好不要出现0x00，因此不能直接写push 0,我们使用寄存器来保存值。代码如下：

#include <stdio.h>

#include <windows.h>

void main(){

LoadLibrary("user32.dll");//加载user32.dll

\_asm{

xor ebx,ebx

push ebx //push 0

push 20646C72h //将helloworld压入栈中

push 6F77206Fh

push 6C6C6568h

mov eax, esp

push ebx //push 0

push eax

push eax

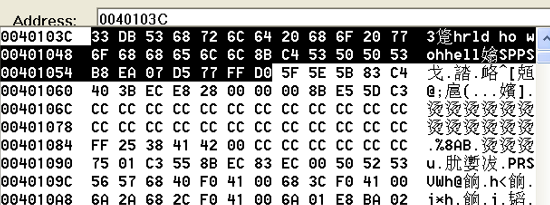
push ebx

mov eax, 77d507eah //77d507eah这个是MessageBox函数在系统中的地址

call eax}

return;}

通过运行程序，添加断点，查看反汇编代码，定位机器码，提取到了shellcode：\x33\xDB\x53\x68\x72\x6C\x64\x20\x68\x6F\x20\x77\x6F\x68\x68\x65\x6C\x6C\x8B\xC4\x53\x50\x50\x53\xB8\xEA\x07\xD5\x77\xFF\xD0



编写完shellcode后，为了确保它能正确运行，需要对其进行编码，并将编码后的结果输出到文件中。本实验采用异或编码的方法，具体是将得到的机器码和指定的一个key值进行异或操作，写入到encode.txt文件中，这利用了异或操作可逆的特性，程序代码如下：

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

void encoder(char\* input, unsigned char key){

int i = 0, len = 0;

FILE \* fp;

len = strlen(input);

unsigned char \* output = (unsigned char \*)malloc(len + 1);

for (i = 0; i<len; i++)

output[i] = input[i] ^ key;

fp = fopen("encode.txt", "w+");

fprintf(fp, "\"");

for (i = 0; i<len; i++){

fprintf(fp, "\\x%0.2x", output[i]);

if ((i + 1) % 16 == 0)

fprintf(fp, "\"\n\"");}

fprintf(fp, "\"");

fclose(fp);

printf("dump the encoded shellcode to encode.txt OK!\n");

free(output);}

int main(){

char sc[] = "\x33\xDB\x53\x68\x72\x6C\x64\x20\x68\x6F\x20\x77\x6F\x68\x68\x65\x6C\x6C\x8B\xC4\x53\x50\x50\x53\xB8\xEA\x07\xD5\x77\xFF\xD0\x90";

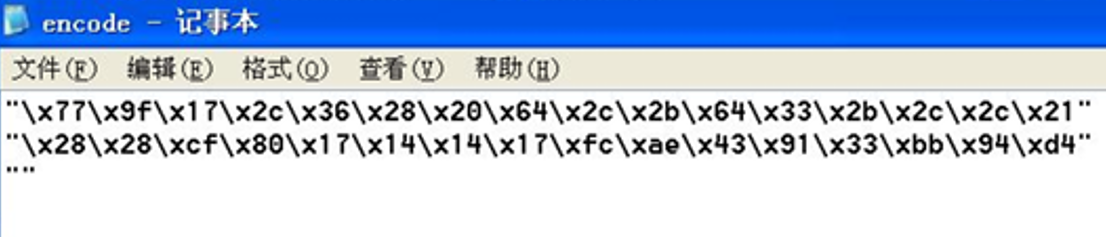
encoder(sc, 0x44);

getchar();

return 0;}

将shellcode代码输入后，得到异或后的shellcode编码：

\x77\x9f\x17\x2c\x36\x28\x20\x64\x2c\x2b\x64\x33\x2b\x2c\x2c\x21\x28\x28\xcf\x80\x17\x14\x14\x17\xfc\xae\x43\x91\x33\xbb\x94\xd4



二进制代码编码后，需要运行解码程序来对编码后的机器码进行解码操作，并且这段解码程序将拼接在待解码程序的前面。因此需要编写一个解码程序，将解码程序的shellcode放在消息框shellcode的前面，把函数的返回地址修改为解码程序shellcode的首地址。

为了能解码消息框shellcode，还需要确定消息框shellcode的位置，因此要先确定解码程序shellcode的长度。代码如下：

#include <stdlib.h>

#include <string.h>

#include <stdio.h>

int main(){

\_\_asm{

call lable;

lable: pop eax;

add eax, 0x15 ;越过decoder记录shellcode起始地址

xor ecx, ecx

decode\_loop:

mov bl, [eax + ecx]

xor bl, 0x44 ;用0x44作为key

mov [eax + ecx], bl

inc ecx

cmp bl, 0x90 ;末尾放一个0x90作为结束符

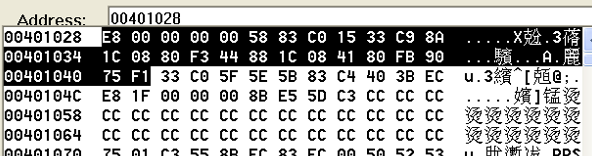
jne decode\_loop }

return 0;}

通过CALL指令，将下一条指令的地址压入栈中，由于pop就是label中的第一条指令，这时通过pop就可以取到当前指令的地址。然后根据得到的机器码的长度，确定出消息框的shellcode在其后面的相对偏移，而这个偏移恰好是15h。通过一个循环来取异或，对代码进行还原，最终解码出原代码。同时，在hello world的shellcode末尾加入了一个0x90字节，因此循环结束的标志就是解码后的字节为0x90。

进入内存区提取得到解码程序的shellcode：

\xE8\x00\x00\x00\x00\x58\x83\xC0\x15\x33\xC9\x8A\x1C\x08\x80\xF3\x44\x88\x1C\x08\x41\x80\xFB\x90\x75\xF1



将编码好的shellcode和解密程序链接起来就得到最终的shellcode：

\xE8\x00\x00\x00\x00\x58\x83\xC0\x15\x33\xC9\x8A\x1C\x08\x80\xF3\x44\x88\x1C\x08\x41\x80\xFB\x90\x75\xF1\x77\x9f\x17\x2c\x36\x28\x20\x64\x2c\x2b\x64\x33\x2b\x2c\x2c\x21\x28\x28\xcf\x80\x17\x14\x14\x17\xfc\xae\x43\x91\x33\xbb\x94\xd4

2.实验验证：

使用示例5-4代码对编码解码代码进行验证，成功：



3.shellcode编码的原理：

Shellcode编码的原理是将原始的机器码进行转换，以适应特定的环境和绕过各种限制。这样做的主要目的是增加Shellcode的稳定性、隐蔽性和免疫力，使其能够成功地执行所需的恶意操作而不被检测或阻止。

不同的系统和应用程序可能使用不同的字符集，某些字符可能被视为坏字符而导致Shellcode执行失败。而且许多安全防护工具会检测并阻止Shellcode的执行。因此需要编码帮助Shellcode绕过这些检测。通过编码，Shellcode的特征会发生改变，使其更难被检测到，从而增加了攻击的成功率。

同时，也需要编写一个解码程序，用于在Shellcode执行前将编码后的Shellcode还原成原始的形式。解码程序通常会被添加到编码后的Shellcode前面，以确保在执行Shellcode之前进行解码。

然而，需要注意的是，即使经过编码，Shellcode仍然可能会被高级的安全工具检测到，因此编码只是增加了Shellcode的免疫力，而不是完全防止被检测。

4.shellcode提取的思想：

提取Shellcode的思想是获取一段被精心设计的机器码，用于执行特定的任务，通常是在计算机系统中进行攻击或执行特定的操作。提取Shellcode的一般思路是：

（1）首先，明确Shellcode的目标功能以及它将在目标系统中执行的方式。

（2）确定所需的动态链接库及其中所需函数的地址，以便在Shellcode中正确调用这些函数。

（3）按照C或汇编的方式编写Shellcode，确保它能够完成所需的功能。在编写时，可能需要考虑到字符集的差异、坏字符的限制等问题。

（4）优化Shellcode：对于直接获得的汇编代码，可能需要进行进一步的优化和加工，以确保其在目标系统中的执行稳定性和免疫性。

（5）编译编写的Shellcode，并获取对应的机器码。这可以通过汇编器或其他工具来完成。

（6）将所得到的机器码组合成最终的Shellcode。

（7）确保提前确定好Shellcode在内存中的地址，并考虑在程序的特定函数中植入Shellcode，并覆盖该函数的返回地址，以触发Shellcode的执行。

总的来说，提取Shellcode的思想是将特定功能的代码转换为机器码，并确保它在目标系统上能够稳定执行，同时尽可能地隐藏和免疫于安全检测。

**四、心得体会：**

通过这次实验，我深刻领悟到了动态链接库加载、函数地址获取以及shellcode的植入与执行等关键概念。

在实验中，我不仅学会了如何提取和编码shellcode，还领悟到了其中的思想和原理。通过编写解码程序，我进一步加深了对汇编语言的理解，提高了自己的编程技能。这次实验让我更加自信地面对复杂的技术挑战，并为未来的学习和工作积累了宝贵的经验。