

《shellcode 编码解码》实验报告

姓名：汤清云 学号：2013536 班级： 1075

实验名称：

Shellcode 编码解码实验。

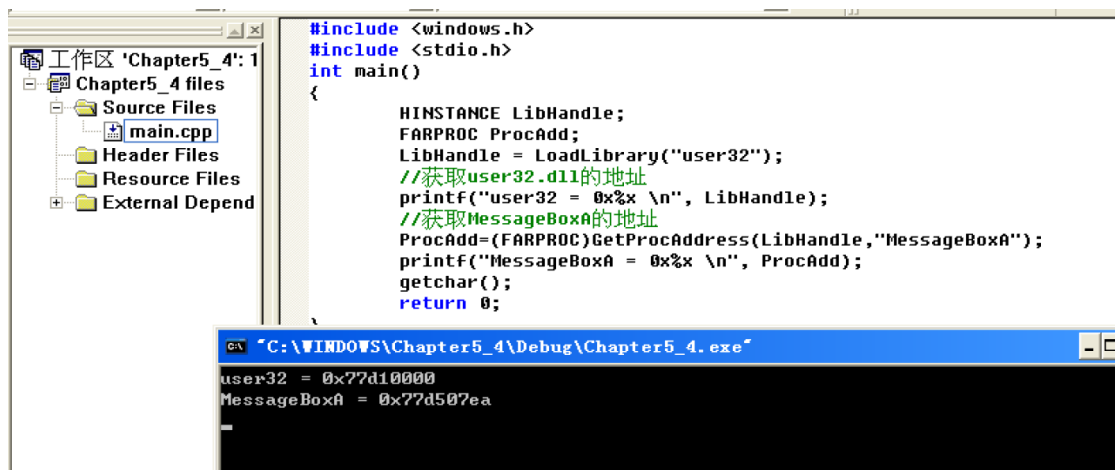
实验要求：

复现第五章实验三（就是能够利用异或解码程序实现弹出对话框的上课讲的内容），并将产生的编码后的 shellcode 在示例 5-4（见上述代码）中进行验证，阐述 shellcode 编码的原理、shellcode 提取的思想。

实验过程：

1. Shellcode 编写方法

第一步：获取 messagebox 函数地址。



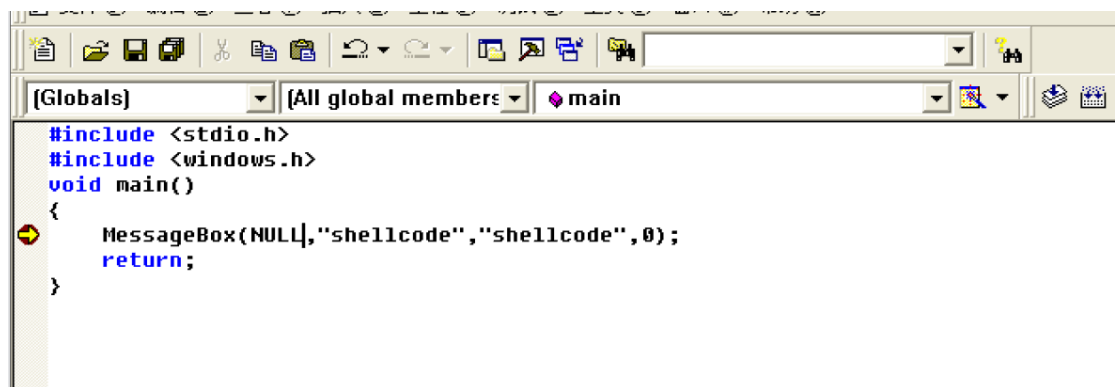
```
#include <windows.h>
#include <stdio.h>
int main()
{
    HINSTANCE LibHandle;
    FARPROC ProcAdd;
    LibHandle = LoadLibrary("user32");
    //获取user32.dll的地址
    printf("user32 = 0x%x \n", LibHandle);
    //获取MessageBoxA的地址
    ProcAdd=(FARPROC)GetProcAddress(LibHandle,"MessageBoxA");
    printf("MessageBoxA = 0x%x \n", ProcAdd);
    getchar();
    return 0;
}
```

C:\WINDOWS\Chapter5_4\Debug\Chapter5_4.exe

user32 = 0x77d10000
MessageBoxA = 0x77d507ea

得到 MessageBoxA 函数在内存中的入口地址：0x77D507EA。

第二步：用 c 语言书写要执行的 shellcode。



```
#include <stdio.h>
#include <windows.h>
void main()
{
    MessageBox(NULL,"shellcode","shellcode",0);
    return;
}
```

找到其对应汇编代码(Go to Disemply)

```

00401019 lea     edi,[ebp-40h]
0040101C mov     ecx,10h
00401021 mov     eax,0CCCCCCCCh
00401026 rep stos dword ptr [edi]
5:      MessageBox(NULL,"shellcode","shellcode",0);
00401028 mov     esi,esp
0040102A push    0
0040102C push    offset string "shellcode" (0042201c)
00401031 push    offset string "shellcode" (0042201c)
00401036 push    0
00401038 call    dword ptr [__imp__MessageBoxA@16 (0042a2ac)]
0040103E cmp     esi,esp
00401040 call    __chkesp (00401070)
6:      return;
7:      }
00401045 nnn     edi

```

00401040 Call CHKESP (00401070)	
地址:	0042201c
0042201C	73 68 65 6C 6C 63 6F 64 65 00 00 00 69 33 38 36 shellcode...i386
0042202C	5C 63 68 68 65 73 70 2E 63 00 00 00 00 00 00 00 \chkesp.c.....
0042203C	54 68 65 20 76 61 6C 75 65 20 6F 66 20 45 53 50 The value of ESP
0042204C	20 77 61 73 20 6E 6F 74 20 70 72 6F 70 65 72 6C was not properl
0042205C	79 20 73 61 76 65 64 20 61 63 72 6F 73 73 20 61 y saved across a

“shellcode” 对应 ascii 码如上。

第三步：换成相应的汇编代码。

```

#include <stdio.h>
#include <windows.h>
void main()
{
    LoadLibrary("user32.dll");//加载user32.dll
    _asm
    {
        xor ebx,ebx
        push ebx//push 0
        push 00000065h//倒序输入shellcode ascii码
        push 646F636Ch
        push 6C656873h
        mov eax,esp

        push ebx
        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;
}

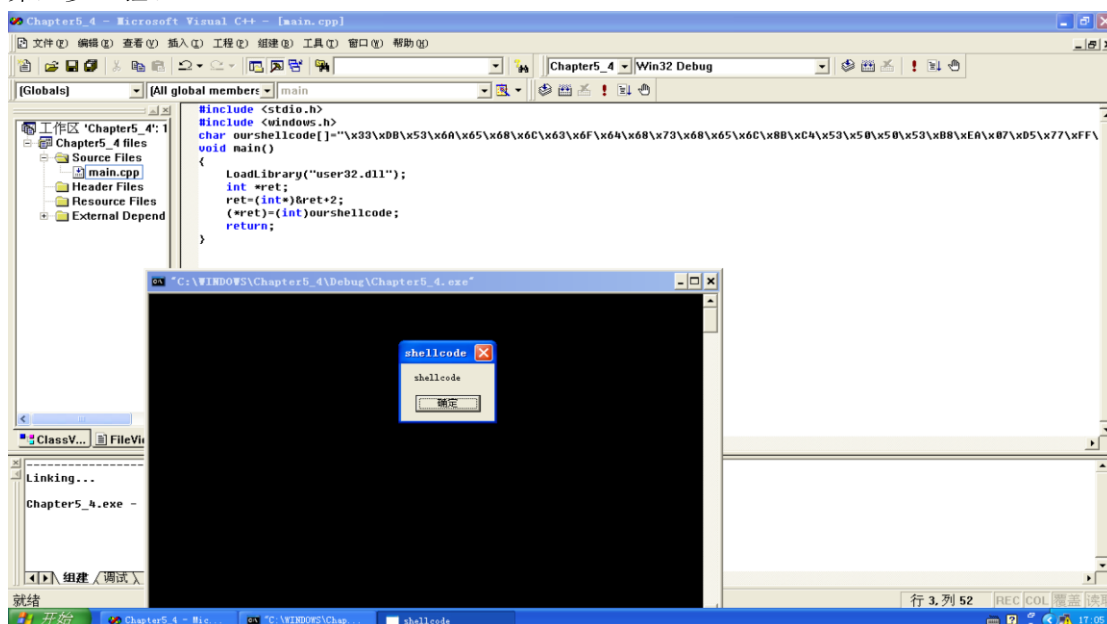
```

第四步：找到对应地址的机器码



\x33\xDB\x53\x6A\x65\x68\x6C\x63\x6F\x64\x68\x73\x68\x65\x6C\x8B\xC4\x53\x50\x50\x53\xB8\xEA\x07\xD5\x77\xFF\xD0

第五步：验证：



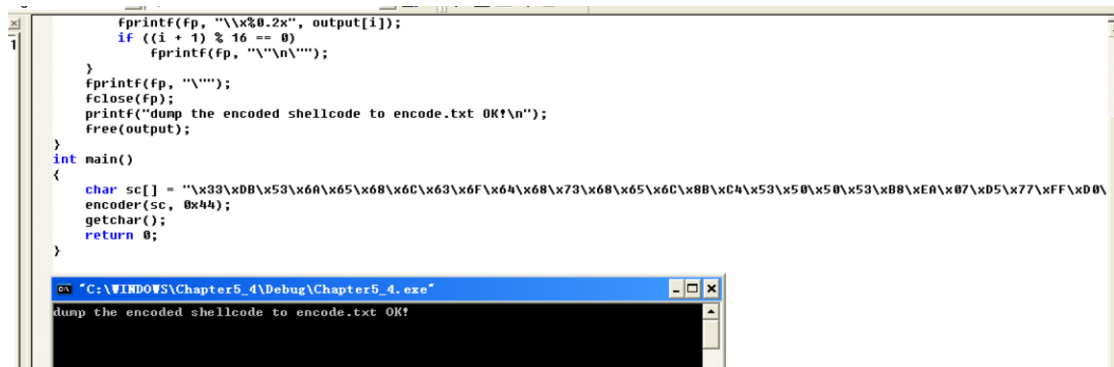
验证成功。

2. Shellcode 编码原理

异或编码是一种简单易用的 shellcode 编码方法，它的编解码程序非常简单。但是，它也存在很多限制，比如在选取编码字节时，不可与已有字节相同，否则会出现 0。此外，还有一些自定义编解码方法被采用，包括简单加解密、alpha_upper 编码、计算编码等。

当 exploit 成功时, shellcode 顶端的解码程序首先运行,它会在内存中将真正的 shellcode 还原成原来的样子,然后执行。这种对 shellcode 编码的方法和软件加壳的原理非常类似。这样,我们只需要专注于几条解码指令,使其符合限制条件就行,相对于直接关注于整段 shellcode 来说使问题简化了很多。

利用 encoder()函数,将之前得到的‘shellcode’的 Shellcode 编码(末尾放一个空指令 0x90 作为结束符)作为参数,它的每一位与 0x44 进行异或运算,对 Shellcode 进行编码,将编码结果写入 encode.txt 中。



The screenshot shows a C program in a text editor and its execution in a command prompt. The C program defines a function `encoder()` that takes a shellcode string and XORs it with 0x44. The `main()` function calls `encoder()` on a specific shellcode string and prints the result to `encode.txt`. The command prompt shows the program running and the message "dump the encoded shellcode to encode.txt OK!".

```
fprintf(fp, "\\x%0.2x", output[i]);
if ((i + 1) % 16 == 0)
    fprintf(fp, "\\n\\n");
}
fprintf(fp, "\\n");
fclose(fp);
printf("dump the encoded shellcode to encode.txt OK!\\n");
free(output);
}

int main()
{
    char sc[] = "\\x33\\x0b\\x53\\x6a\\x65\\x68\\x6c\\x63\\x6f\\x64\\x68\\x73\\x68\\x65\\x6c\\x8b\\xc4\\x53\\x50\\x50\\x53\\x88\\xe8\\x07\\x05\\x77\\xff\\xd0\\x14\\x14\\x17\\xfc\\xae\\x43\\x91\\x33\\xbb\\x94\\xd4";
    encoder(sc, 0x44);
    getchar();
    return 0;
}
```

```
C:\WINDOWS\Chapter5_4\Debug\Chapter5_4.exe
dump the encoded shellcode to encode.txt OK!
```

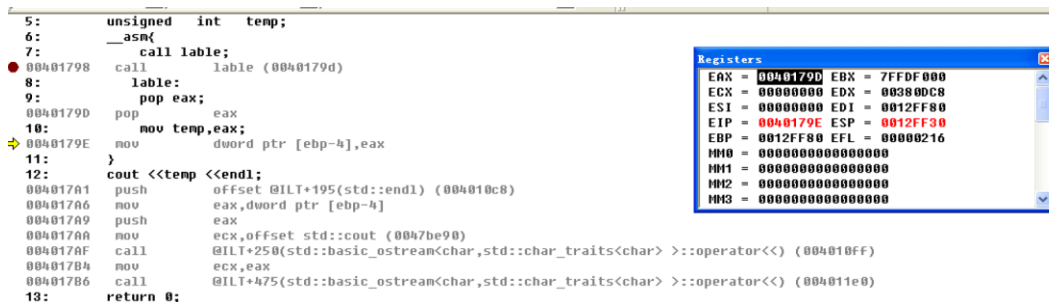
得到密码为:

"\x77\x9f\x17\x2e\x21\x2c\x28\x27\x2b\x20\x2c\x37\x2c\x21\x28\xcf\x80\x17\x14\x14\x17\xfc\xae\x43\x91\x33\xbb\x94\xd4"

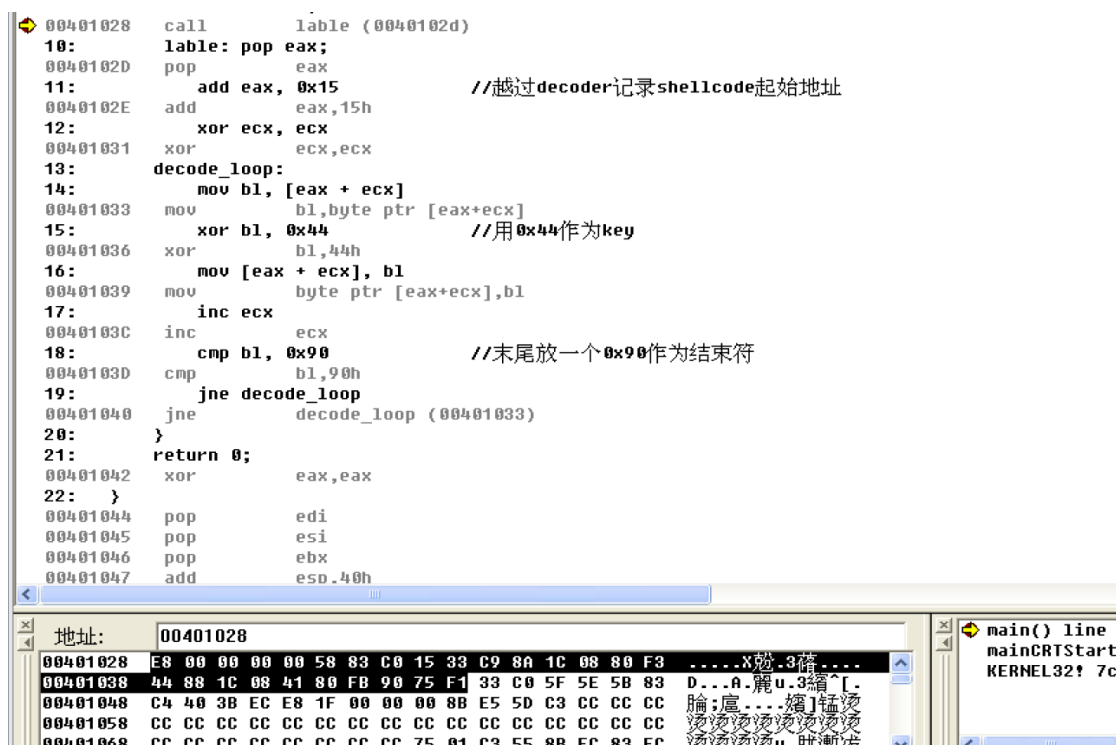


3. shellcode 解码原理

call lable; lable: pop eax;"之后, eax 的值就是当前指令地址了。原因是 call lable 的时候,会将当前 EIP 的值(也就是下一条指令 pop eax 的指令地址)入栈。因此,我们通过下面的程序来产生含有解码程序的 Shellcode



将每次把 shellcode 的代码异或特定 key (0x44) 后重新覆盖原先 shellcode 的代码。末尾，放一个空指令 0x90 作为结束符。再从 call label 语句的地址找到我们所需的答案：



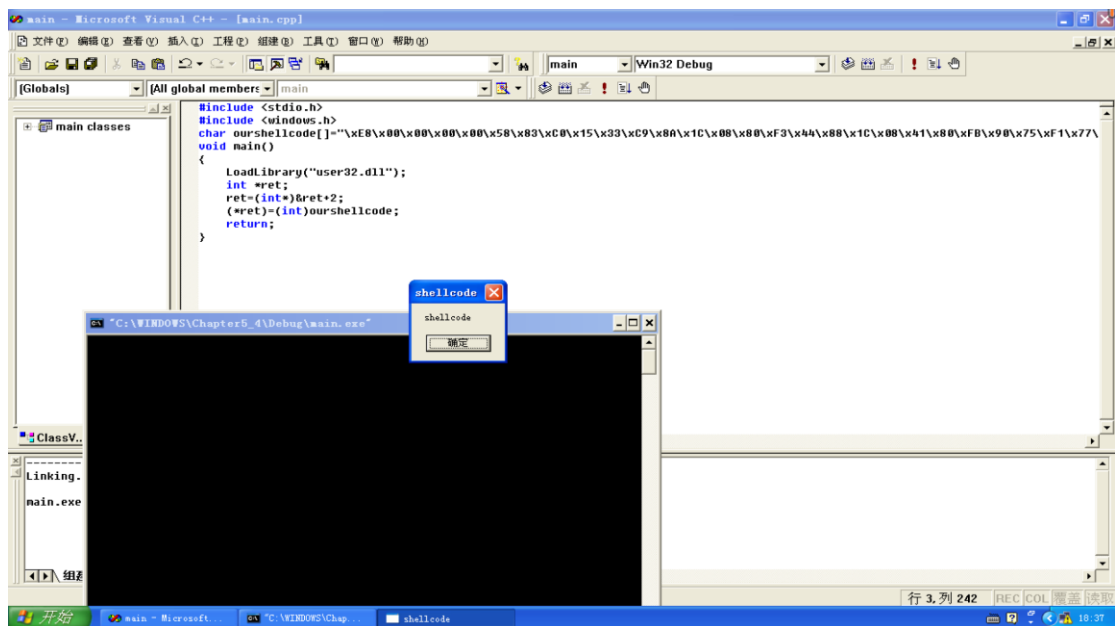
即

“\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”.

再将其与之前调用 MessageBoxA 输出“shellcode”的 Shellcode 的编码连接起来有：

\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\x2e\x21\x2c\x28\x27\x2b\x20\x2c\x37\x2c\x21\x28\xcf\x80\x17\x14\x14\x17\xfc\xae\x43\x91\x33\xbb\x94\xd4

再次测试结果为：



完成实验。

4. 总结

shellcode 编写思路：

- 1) 装载动态链接库 user32.dll
- 2) 获得 MessageBoxA 函数的入口地址
- 3) c 语言书写要执行的 shellcode
- 4) 找到其对应的汇编代码，进行加工，并写入项目中
- 5) 根据汇编代码，找到对应地址中的机器码，编写 shellcode
- 6) 进行 shellcode 利用验证

shellcode 编码原理：

对于二进制 Shellcode 机器代码的编码，通常采用类似“加壳”思想的手段，采用自定义编码的方法完成 shellcode 的编码，同时通过精心构造精简干练的解码程序，放在 shellcode 开始执行的地方，完成 shellcode 的编解码。

- 1) 在单独程序中编写编码代码，将原始 shellcode 进行异或编码。
- 2) 编写解码代码，先定位到 shellcode 起始位置（含解码代码），再向后偏移到执行 shellcode 的地方，将编码后的 shellcode 与原字符进行异或，实现解码。
- 3) 将上步的解码代码的机器码与编码后的 shellcode 进行链接，形成完整的 shellcode 代码
- 4) 进行 shellcode 利用验证

心得体会：

学习了 shellcode 的编码以及解码过程，对漏洞利用有了进一步了解。