



南開大學
Nankai University

计算机学院
软件安全实验报告

实验五：Shellcode 编码解码

姓名：林盛森

学号：2312631

专业：计算机科学与技术

2025 年 4 月 14 日

目录

1 实验名称	2
2 实验要求	2
3 实验过程	2
3.1 实验一（刚开始做错了实验，以为做实验一，实验三在后面，这个是实验一）	2
3.2 shellcode 的编写	5
3.3 shellcode 的编码	6
3.4 shellcode 的解码	8
4 心得体会	10

1 实验名称

Shellcode 编码解码

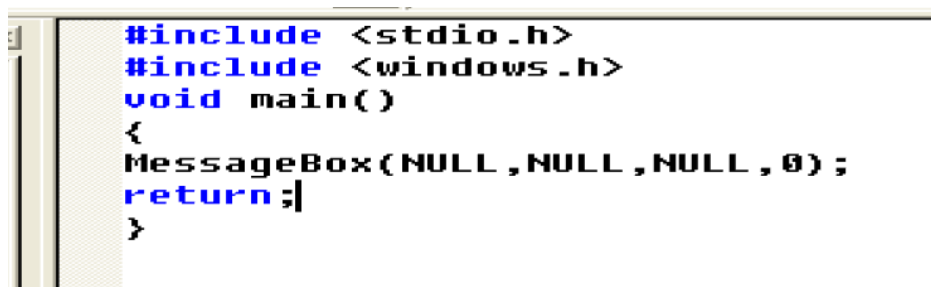
2 实验要求

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。

3 实验过程

3.1 实验一（刚开始做错了实验，以为做实验一，实验三在后面，这个是实验一）

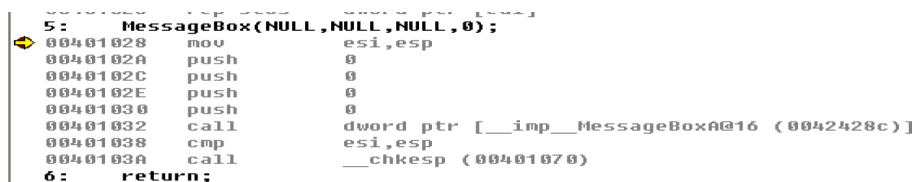
首先输入实验代码，执行后发现正常运行。



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

图 3.1: Enter Caption

之后，我们设置断点，运行代码之后，点击 go to disassembly 进行反汇编，获得汇编代码。



```
5:      MessageBox(NULL, NULL, NULL, 0);
00401028  mov     esi, esp
0040102A  push    0
0040102C  push    0
0040102E  push    0
00401030  push    0
00401032  call    dword ptr [__imp__MessageBoxA@16 (0042428c)]
00401038  cmp     esi, esp
0040103A  call    __chkesp (00401070)
6:      return;
```

图 3.2: Enter Caption

我们现在需要获取 MessageBox 函数的地址，在这里采取动态获取地址的方法：

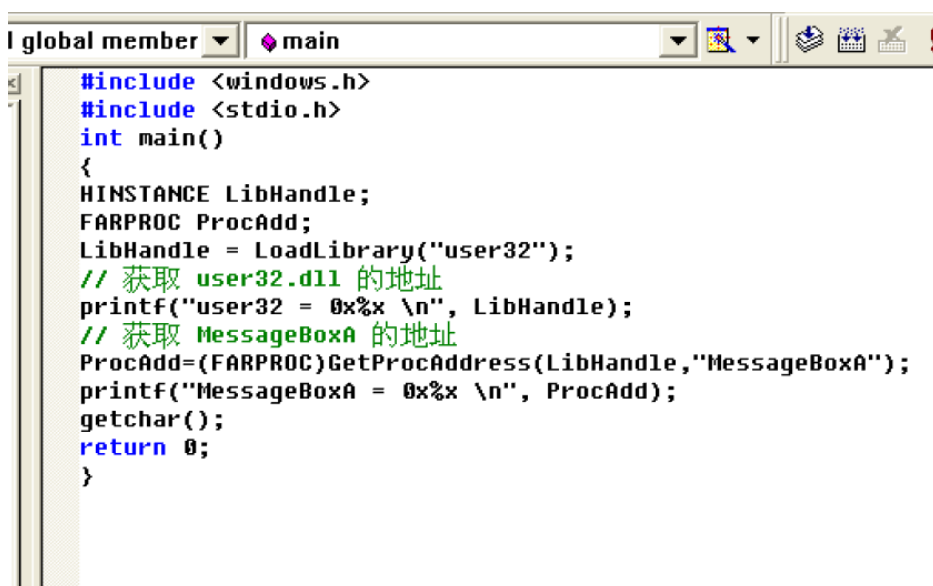


图 3.3: Enter Caption

执行之后我们得到 MessageBox 函数的入口地址为: 0x77d507ea。

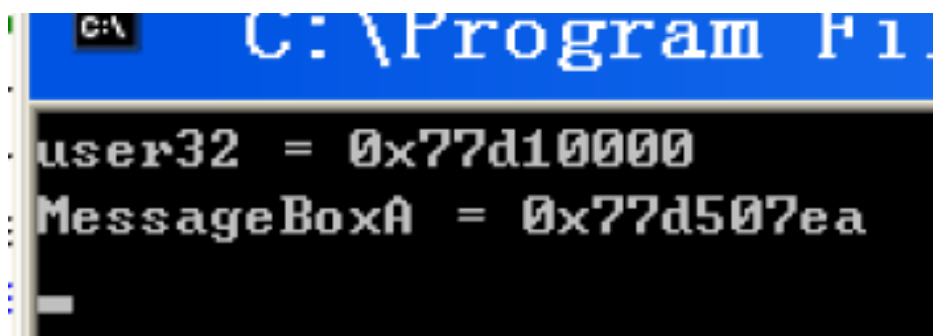
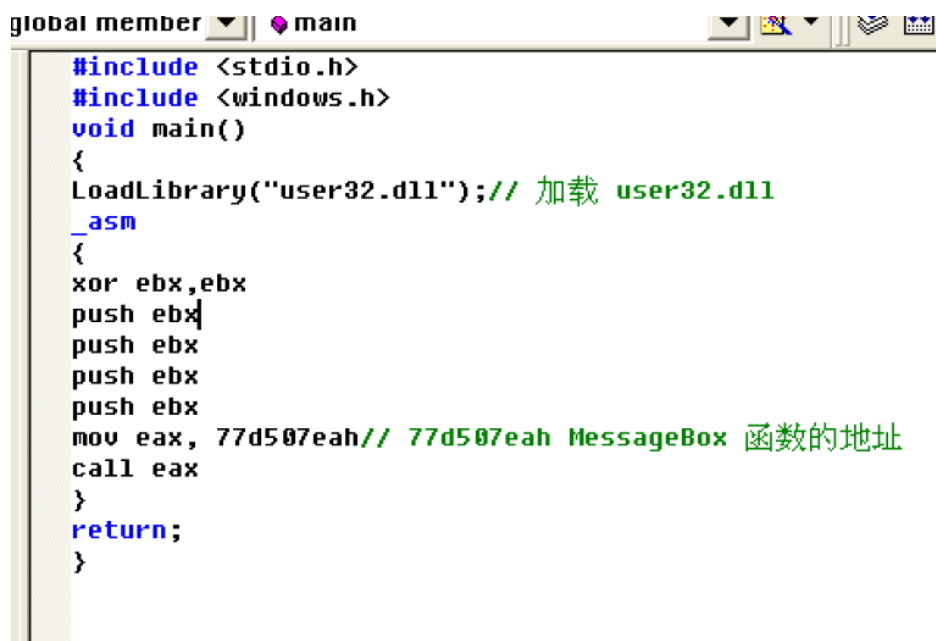


图 3.4: Enter Caption

由于 shellcode 编码时有需要绕过坏字符的限制, 我们不能直接 push0, 所以对代码进行一些修改, 并且将改写后的代码放在 __asm 中, 这样我们就可以在 c 语言中插入汇编代码。如下:



```
global member
main
#include <stdio.h>
#include <windows.h>
void main()
{
    LoadLibrary("user32.dll");// 加载 user32.dll
    _asm
    {
        xor ebx,ebx
        push ebx
        push ebx
        push ebx
        mov eax, 77d507eah// 77d507eah MessageBox 函数的地址
        call eax
    }
    return;
}
```

图 3.5: Enter Caption

通过 `xor ebx, ebx`, 让 `ebx` 与自身异或, 既实现让 `ebx` 清零, 这样就避免了使用 `push 0` 去传递参数, 否则读机器码读到 0 的时候程序就会停止。接着运行程序, 发现成功出现错误的弹窗, 说明修改的代码是正确的。

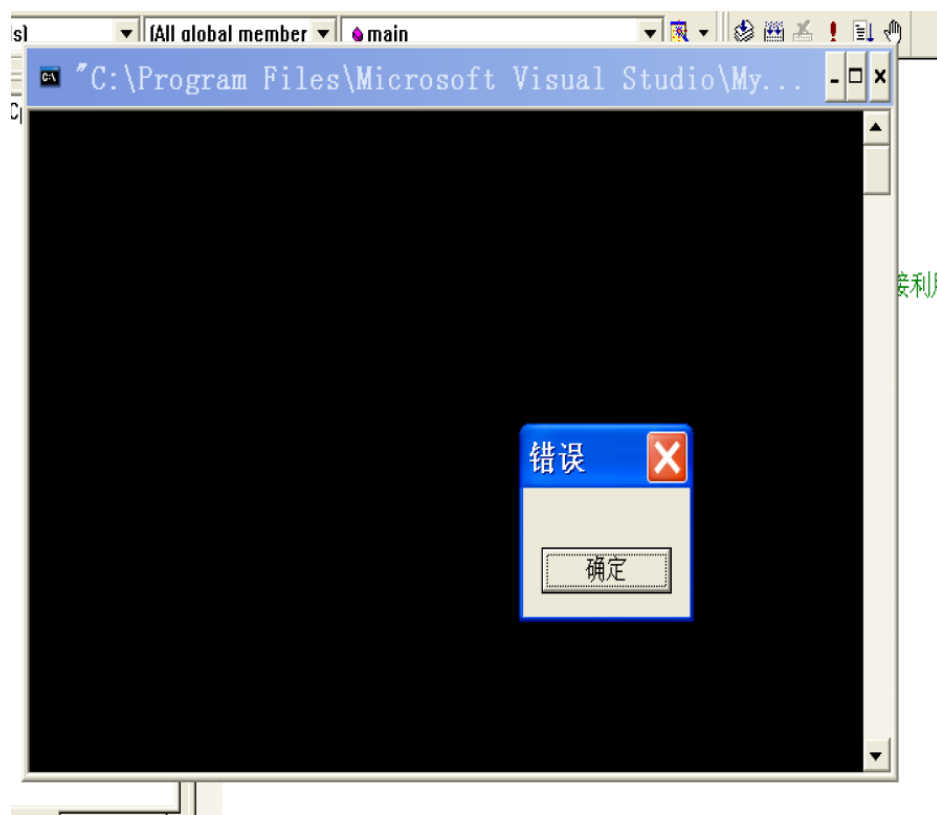


图 3.6: Enter Caption

接着我们需要获得这段程序的机器码, 进入反汇编, 通过跳转代码开始的地址, 查看中的内容, 得

到对应的机器码应该是：33 DB 53 53 53 B8 EA 07 D5 77 FF D0。

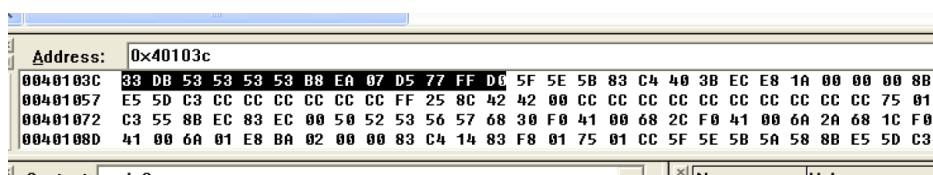


图 3.7: Enter Caption

然后把得到的机器码放在我们想要插入的 shellcode 中，执行如下程序：

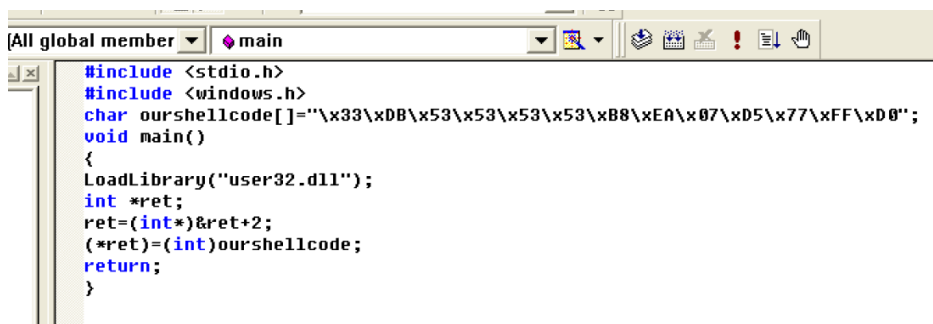


图 3.8: Enter Caption

首先定义了一个 ret 指针,ret=(int*)&ret+2 的作用是让 ret 指向返回地址,(*ret)=(int)ourshellcode 的作用是把返回地址改成 ourshellcode 的起始地址,这样当主函数结束时就会跳转去执行 ourshellcode 中机器码所代表的行为。运行程序之后发现确实显示了弹窗,说明 shellcode 的编写即机器码的提取是正确的。

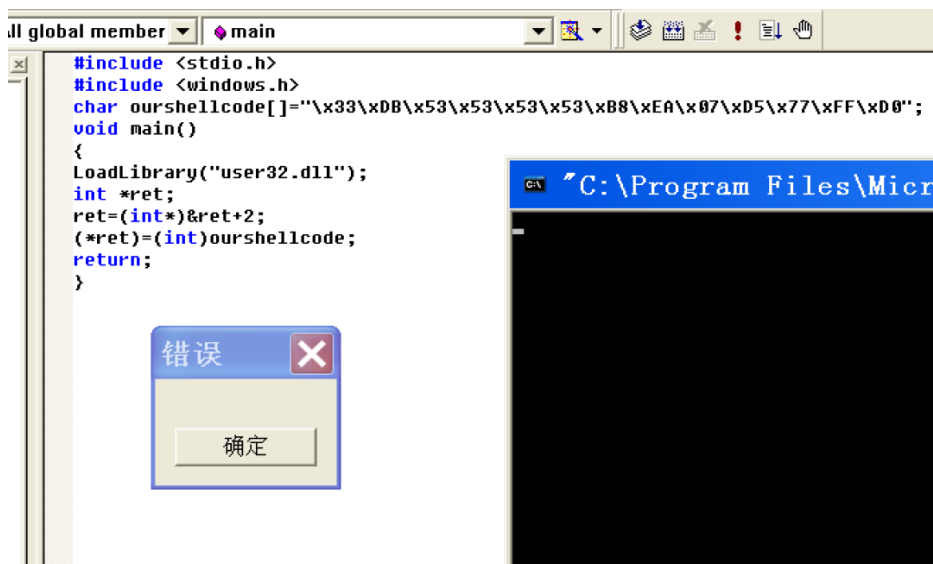


图 3.9: Enter Caption

3.2 shellcode 的编写

接着我们来实现弹窗 “hello world” 的 shellcode 的编写。“hello world” ASCII 码为:

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

但计算机中数据的存储是小端序存储的，所以我们要将“hello world”对应的 ASCII 码的顺序给倒过来，我们执行下面的程序，可以看到弹出了“hello world”的窗口。



图 3.10: Enter Caption

接着我们需要进入反汇编得到这段代码的机器码。对应的地址是从 0040103C 开始到 0040105A:

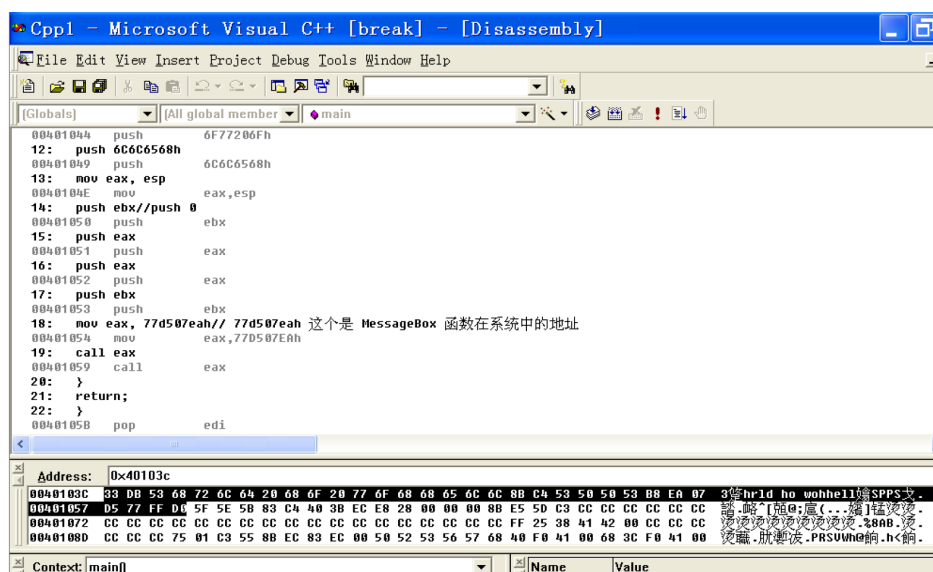


图 3.11: Enter Caption

提取出的 shellcode 代码就是:

\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\xFF\xD0

3.3 shellcode 的编码

接下来我们对得到的 shellcode 进行编码，使用异或编码:

```
1 #include <stdlib.h>
2 #include <string.h>
3 #include <stdio.h>
4 void encoder(char* input, unsigned char key)
5 {
6     int i = 0, len = 0;
7     FILE* fp;
8     len = strlen(input);
9     unsigned char* output = (unsigned char*)malloc(len + 1);
10    for (i = 0; i < len; i++)
11        output[i] = input[i] ^ key;
12    fp = fopen("encode.txt", "w+");
13    fprintf(fp, "\\");
14    for (i = 0; i < len; i++)
15    {
16        fprintf(fp, "\\x%0.2x", output[i]);
17        if ((i + 1) % 16 == 0)
18            fprintf(fp, "\\n\\n");
19    }
20    fprintf(fp, "\\");
21    fclose(fp);
22    printf("dump the encoded shellcode to encode.txt OK!\\n");
23    free(output);
24 }
25 int main()
26 {
27     char sc[] =
28         "\\x33\\xDB\\x53\\x68\\x72\\x6C\\x64\\x20\\x68\\x6F\\x20\\x77\\x6F\\x68\\x68\\x65\\x6C\\x6C\\x8B
29         \\xC4\\x53\\x50\\x50\\x53\\xB8\\xEA\\x07\\xD5\\x77\\xFF\\xD0\\x90";
30     encoder(sc, 0x44);
31     getchar();
32     return 0;
33 }
```

这段代码的逻辑就是，把传入的机器码每一项都与 0x44 相异或，从而得到编码之后的 shellcode 代码。注意我们在末尾放了一个 x90 的结束符。

执行之后在项目中生成了一个文本文件，其中存储着 shellcode 的编码。


```

{
    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%.2x", output[i]);
        if ((i + 1) % 16 == 0)
            fprintf(fp, "\\n\\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"
        encoder(sc, 0x44);
    getchar();
    return 0;
}

```

C:\Program Files\Microsoft Visual Stu
dump the encoded shellcode to encode.txt OK!

图 3.12: Enter Caption

encode - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```

""\\x77\\x9F\\x17\\x2c\\x36\\x28\\x20\\x64\\x2c\\x2b\\x64\\x33\\x2b\\x2c\\x21"
""\\x28\\x28\\xcf\\x80\\x17\\x14\\x14\\x17\\xfc\\xae\\x43\\x91\\x33\\xbb\\x94\\xd4"
....

```

图 3.13: Enter Caption

3.4 shellcode 的解码

为了能让 shellcode 正确运行，我们还需要获得解码器的机器码，所生成的解码程序对应的机器码与编码后的 shellcode 联合执行，才能保证程序的正确性。通过以下代码进行解码：

```

1 #include <stdlib.h>
2 #include <string.h>
3 #include <stdio.h>
4
5 int main()
6 {
7     __asm
8     {

```

```

9      call lable;
10     lable: pop eax;
11         add eax, 0x15          ;越过decoder记录shellcode起始地址
12         xor ecx, ecx
13     decode_loop:
14         mov bl, [eax + ecx]
15         xor bl, 0x44          ;用0x44作为key
16         mov [eax + ecx], bl
17         inc ecx
18         cmp bl, 0x90          ;末尾放一个0x90作为结束符
19         jne decode_loop
20     }
21     return 0;
22 }

```

这段代码的逻辑是，在循环中，每次取编码后的 shellcode 代码对其进行异或操作来实现解码的操作，而循环的终止语句就是判断是否等于我们之前存入的结束符 0x90，如果相等就说明已经解码完毕。

问题的关键是如何定位到 shellcode，核心语句在于“call lable; lable: pop eax;”。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，然后当执行 pop eax 的时候，就把 eip 的值给了 eax 寄存器，所以只要对 eax 寄存器加上解码器的长度，就可以定位到 shellcode。并且由于解码器应该是放在最终 shellcode 的低字节，所以要找到 shellcode 应该用 add 指令而不是 sub。

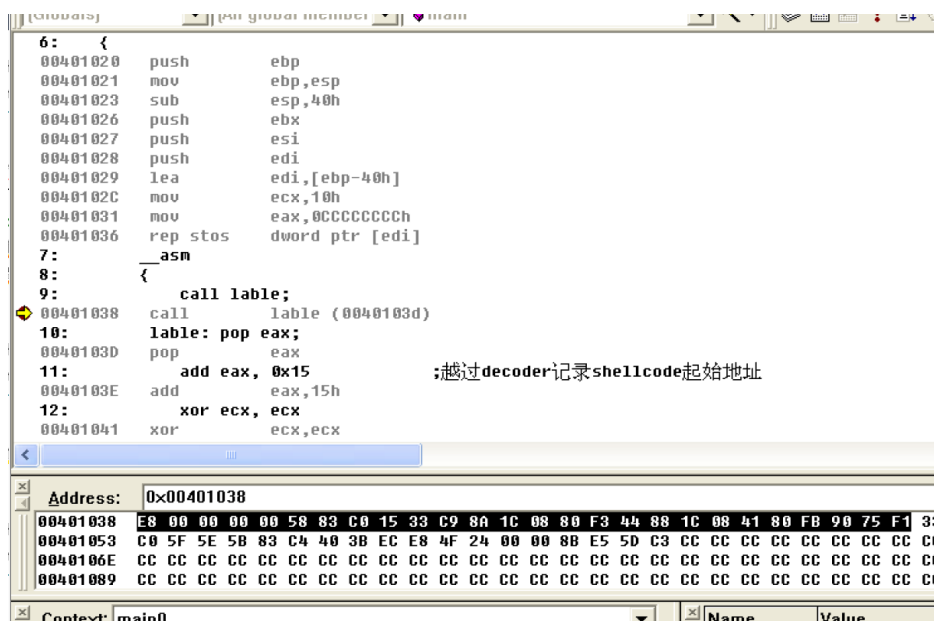


图 3.14: Enter Caption

再通过之前提取机器码的操作，我们得到了解码器的机器码为：E8 00 00 00 00 58 83 C0 15 33 C9 8A 1C 08 80 F3 44 88 1C 08 41 80 FB 90 75 F1

然后我们只需把解码器的机器码和编码后的 shellcode 拼接起来，即可得到完整的 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

通过 5-4 示例验证, 完全正确。

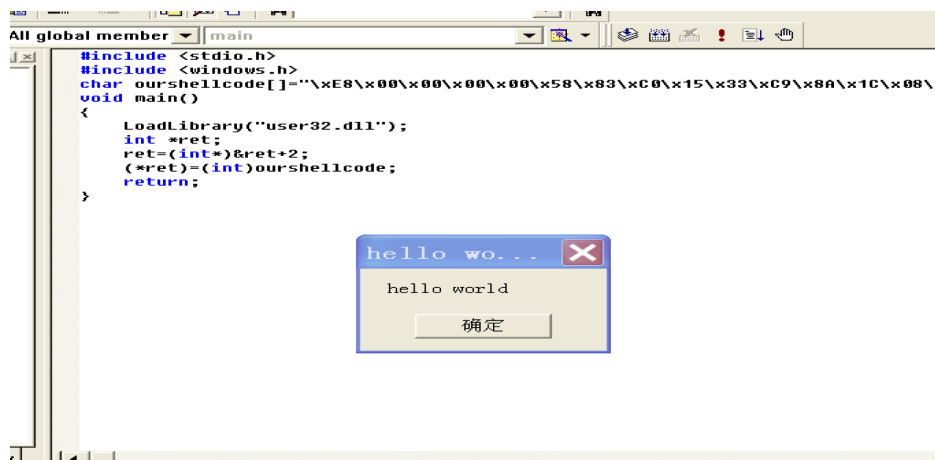


图 3.15: Enter Caption

4 心得体会

1. 了解了如何去提取 shellcode、对 shellcode 编码、解码。
2. shellcode 提取的原理就是, 先获得我们想要执行程序的汇编代码, 然后对其进行一些修改, 然后可以在 c 语言通过 `_asm` 中插入汇编代码, 再进行反汇编, 从而在代码对应的地址处获得机器码, 从而完成了 shellcode 的提取。
3. shellcode 编写的原理跟提取的类似, 都是写出汇编代码, 注意汇编代码中可能需要填入一些函数的地址, 对于 dll 库中的函数我们需要动态获得地址, 然后反汇编找到机器码, 就完成了编写。
4. shellcode 的编码采用异或编码, 将每个机器码与某一个值异或即可完成编码, 解码程序也要放在 shellcode 中, 完成对 shellcode 的解码, 并且要注意解码程序如何定位到 shellcode 的起始位置。