《漏洞利用及渗透测试基础》实验报告

姓名：张洋　　学号：2111460　　班级：信安二班

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 5-4 中进行验证，阐述 shellcode 编码的原理、shellcode 提取的思想。

实验三：在实验二基础上，对 shellcode 进行编码后再进行利用。

实验二：在实验一基础上，自己编写调用 Messagebox 输出“hello world”的 Shellcode，并进行利用测试。

实验一：基于示例 5-1，向其植入一段代码，弹出 MessageBox 窗体。Windows XP 环境下，基于 VC6 进行实验。

实验过程：

1. ShellCode 编写

由于 Shellcode 必须以机器码的形式存在，因此，本实验的关键点在于如何得到机器代码。实验具体的操作步骤如下：

第一步：获得函数入口地址

main.cpp *

```
#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;
}
```

运行后得到 MessageBoxA 函数在内存中的入口地址 0x77D507EA。

```
"C:\Program Files\Micr
user32 = 0x77d10000
MessageBoxA = 0x77d507ea
```

第二步：书写要执行的 shellcode

main.cpp

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

找到对应的汇编代码：

```
5:        MessageBox(NULL,"hello world","hello world",0);
00401028  mov         esi,esp
0040102A  push        0
0040102C  push        offset string "hello world" (0041f01c)
00401031  push        offset string "hello world" (0041f01c)
00401036  push        0
00401038  call        dword ptr [__imp__MessageBoxA@16 (0042428c)]
0040103E  cmp         esi,esp
00401040  call        __chkesp (00401070)
6:        return 0;
```

找到“hello world”对应的 ascii 码。

```
Address:  0041f01c
0041F01C  68 65 6C 6C 6F 20 77 6F 72 6C 64
```

“hello world”对应 ascii 码为 68 65 6C 6C 6F 20 77 6F 72 6C 64。

第三步：换成对应的汇编代码

```
#include <windows.h>
#include <stdio.h>
int main()
{
    LoadLibrary("user32.dll");//加载user32.dll
    __asm{
        xor ebx,ebx
        push ebx
        push 00646C72h
        push 6F77206Fh
        push 6C6C6568h
        mov eax,esp
        push ebx
        push eax
        push eax
        push ebx
        mov eax,0x77D507EA
        call eax
    }
    return 0;
}
```

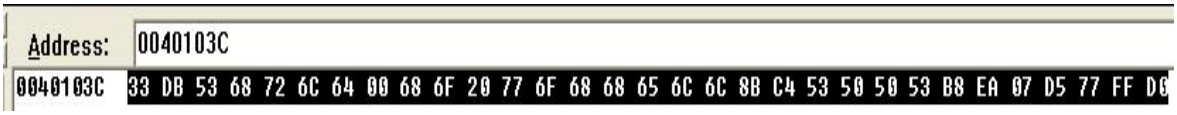
汇编代码解释：

```
xor ebx,ebx        ; 将 ebx 的值设置为 0
push ebx           ; 将 ebx 的值入栈
push 00646C72h     ; 倒序输入 hello world 的 ascii 码
push 6F77206Fh
push 6C6C6568h
mov eax,esp        ; 将栈顶指针存入 eax（栈顶指针的值就是字符串的首地址）
push ebx           ; 入栈 Messagebox 的 4 个参数-类型
push eax           ; 入栈 Messagebox 的 4 个参数-标题
push eax           ; 入栈 Messagebox 的 4 个参数-消息
push ebx           ; 入栈 Messagebox 的 4 个参数-句柄
mov eax,0x77D507EA
call eax           ; 调用 MessageBoxA 函数
```

第四步：找到对应的机器码

```
6:        __asm{
7:           xor ebx,ebx
0040103C  xor         ebx,ebx
8:           push ebx
0040103E  push        ebx
9:           push 00646C72h
0040103F  push        646C72h
10:          push 6F77206Fh
00401044  push        6F77206Fh
11:          push 6C6C6568h
00401049  push        6C6C6568h
12:          mov eax,esp
0040104E  mov         eax,esp
13:          push ebx
00401050  push        ebx
14:          push eax
00401051  push        eax
15:          push eax
00401052  push        eax
16:          push ebx
00401053  push        ebx
17:          mov eax,0x77D507EA
00401054  mov         eax,77D507EAh
18:          call eax
00401059  call        eax
19:       }
```

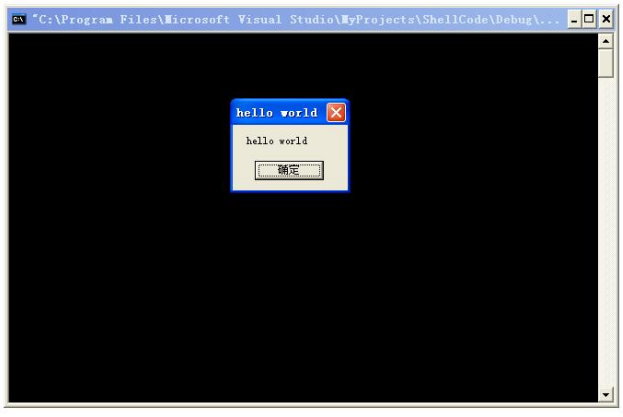
从 0040103C 到 0040105A 对应的机器码如下图：



得到的 shellcode 为：33 DB 53 68 72 6C 64 00 68 6F 20 77 6F 68 68 65 6C 6C 8B C4 53 50 50 53 B8 EA 07 D5 77 FF D0

第五步：验证 shellcode

```
#include <windows.h>
#include <stdio.h>
char ourshellcode[] = "\x33\xDB\x53\x68\x72\x6C\x64\x00\x68\x6F\x20\x77\x6F\x68\x68\x65\x6C\x6C\x8B\xC4\x53\x50\x50\x53\xB8\xEA\x07\xD5\x77\xFF\xD0";
int main()
{
    LoadLibrary("user32.dll");
    int *ret;
    ret=(int*)&ret+2;
    (*ret)=(int)ourshellcode;
    return 0;
}
```



弹出窗体并输出“hello world”，验证成功。

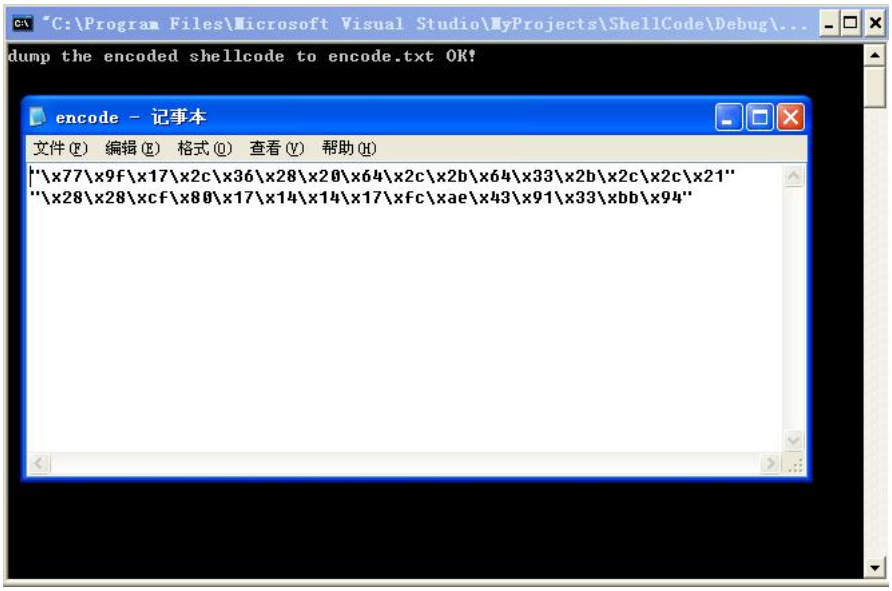
2. Shellcode 编码

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 编码作为参数，让它的每一位与 0x44 进行异或运算，对 Shellcode 进行编码，将编码结果写入 encode.txt 中。

```
#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";
    encoder(sc, 0x44);
    getchar();
    return 0;
}
```



编码后的 shellcode 为:“\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”

3. Shellcode 解码

```
#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;
}
```

“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。之后的代码将每次将 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。

```
  8:    call lable;
00401038   call        lable (0040103d)
  9:     lable: pop eax;
0040103D   pop         eax
 10:    add eax, 0x15 ;越过 decoder 记录 shellcode 起始地址
0040103E   add         eax,15h
 11:    xor ecx, ecx
00401041   xor         ecx,ecx
 12:    decode_loop:
 13:    mov bl, [eax + ecx]
00401043   mov         bl,byte ptr [eax+ecx]
 14:    xor bl, 0x44 ;用 0x44 作为 key
00401046   xor         bl,44h
 15:    mov [eax + ecx], bl
00401049   mov         byte ptr [eax+ecx],bl
 16:    inc ecx
0040104C   inc         ecx
 17:    cmp bl, 0x90 ;末尾放一个 0x90 作为结束符
0040104D   cmp         bl,90h
 18:    jne decode_loop
00401050   jne         decode_loop (00401043)
 19:    }
 20:   return 0;
00401052   xor         eax,eax
 21:    }
```

```
Address:  00401038
00401038  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 33 C0 5F
0040106D  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010A2  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010D7  CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
```

提取得的机器码为：

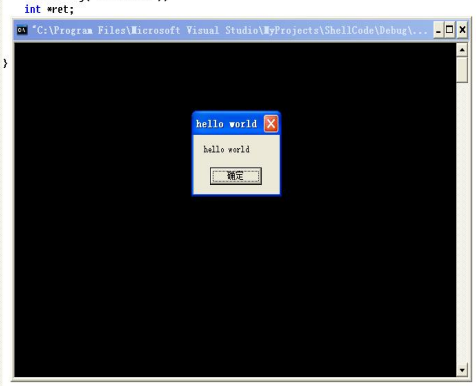
“\xE8\x00\x00\x00\x00\x58\x83\xC0\x15\x33\xC9\x8A\x1C\x08\x80\xF3\x44\x88\x1C\x08\x41\x80\xFB\x90\x75\xF1\x33”

链接两段机器码后，得到完整 shellcode 如下：

“\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\x33\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”

再次测试结果为：

```
#include <windows.h>
#include <stdio.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\x33\x77\x9F\x17\x2c\x36\x28\x20\x64\x2c\x2b\x64\x33\x2b\x2c\x2c\x21\x28\x
int main()
{
    LoadLibrary("user32.dll");
    int *ret;
```




```
}
```

实现了解码。

心得体会：

通过本次实验，我更加深入地了解了 shellcode 编写，编码和解码地步骤和原理。以下是我做的一些总结：

在编写 shellcode 时，需要考虑一些限制和约束，如代码长度、CPU 架构、系统调用等。

编写 Shellcode 的一般步骤如下：

1. 确定攻击目标和利用方式；
2. 编写原始 Shellcode，包括汇编指令、系统调用等；
3. 对 Shellcode 进行编码，以便绕过防御机制和字符过滤等；
4. 将编码后的 Shellcode 插入到攻击载荷中，使其能够被成功执行。

在编写 Shellcode 时，为了避免被防御机制拦截，需要进行编码和解码操作。常用的编码方式有十六进制编码、Base64 编码、ROT13 编码等。常见的解码方式有手动解码和自动解码两种。

手动解码是指使用调试器等工具，逐步执行 Shellcode 中的指令，并将其转化为可读的字符串。自动解码则是使用专门的工具，如 Metasploit Framework、Veil-Evasion 等，自动化地将编码后的 Shellcode 进行解码，并生成可用的攻击载荷。

编码和解码的原理主要是利用特定的算法和转换方式，将原始 Shellcode 转化为另一种形式，以达到绕过字符过滤和防御机制的目的。在解码时，需要逆向编码过程，将编码后的 Shellcode 还原为原始的二进制代码。

除此之外，在实验过程中，我还遇到了一些问题。在编码的过程中，由于 shellcode 的编码中存在 0x00，会发生自动截断，即遇到 0x00 会停止编码。通过手动修改可以避免自动截断的发生，完成实验内容。