《漏洞利用及渗透测试基础》实验报告

姓名：田晋宇　　学号：2212039　　班级：物联网班

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 5-1 中进行验证，阐述 shellcode 编码的原理，shellcode 提取的思想。

实验过程：

Shellcode 编码解码思想：

由于在漏洞发现之处并不会给出完整的 Shellcode，因此我们需要对 Shellcode 进行编码，对一些特殊字符需要进行转码，比如对于 strcpy 等函数造成的缓冲区溢出，会认为 NULL 是字符串的终结，所以 shellcode 编码中不能有 NULL，常见的一种编码方式为异或编码，即 xor a，b，a 为待编码的 shellcode，key 为自定义，该 key 值使得编码后的 shellcode 不会出现 0。

对 shellcode 编码之后意味着要进行解码，一般来说，当前指令知道自己的位置，这样就可以计算出编码的 shellcode 的位置开始解码，我们使用 call 指令的方法来获取 shellcode 编码的起始位置。具体思想如下：jmp 指令跳转到 call 指令，该 call 指令位于编码 shellcode 之前，call 指令会创建一个新的栈，因此将当前的 EIP 指针压栈，即为当前的 shellcode 起始地址，call 调用的过程中将压栈的地址弹出保存到寄存器中，利用保存的寄存器进行 shellcode 解码，jmp 到 shellcode 处执行。

具体实验操作：

在编码程序的主函数中，我们选定了已经生成的 textmessage 的 shellcode 编码，通过 encode 函数对其进行编码，选定的 key 值为 0x44，这样的 key 值可以使编码后的 shellcode 没有 0 的出现。

```
int main()
{
    char sc[] =  "\x33\xDB\x53\x68\x72\x6C\x64\x20\x68\x6F\x20\x77\x6F\x68\x68\x65\x6C\x6C\x8E
    encoder(sc, 0x44);
    getchar();
    return 0;
}
```

在编码函数中我们将 input 的每一位与 key 值进行异或，并赋值给 output，最终将编码结果写入 encode.txt 文件中。

```
#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);
}
```

获得的编码后的 shellcode 如下：

接下来我们对编码后的 shellcode 进行解码，以下为解码程序，add eax，0x14 用来获取接下来 shellcode 的起始地址，eax 为当前指令的地址，ecx 初始值设为 0，作为循环的偏移量，循环的判断条件为获取的一位 shellcode 代码是否为 0x90，作为终止判断条件。

```
void main()
{
    __asm
    {
        add eax, 0x14
        xor ecx, ecx
    decode_loop:
        mov bl, [eax + ecx]
        xor bl, 0x44
        mov [eax + ecx], bl
        inc ecx
        cmp bl, 0x90
        jne decode_loop
    }
}
```

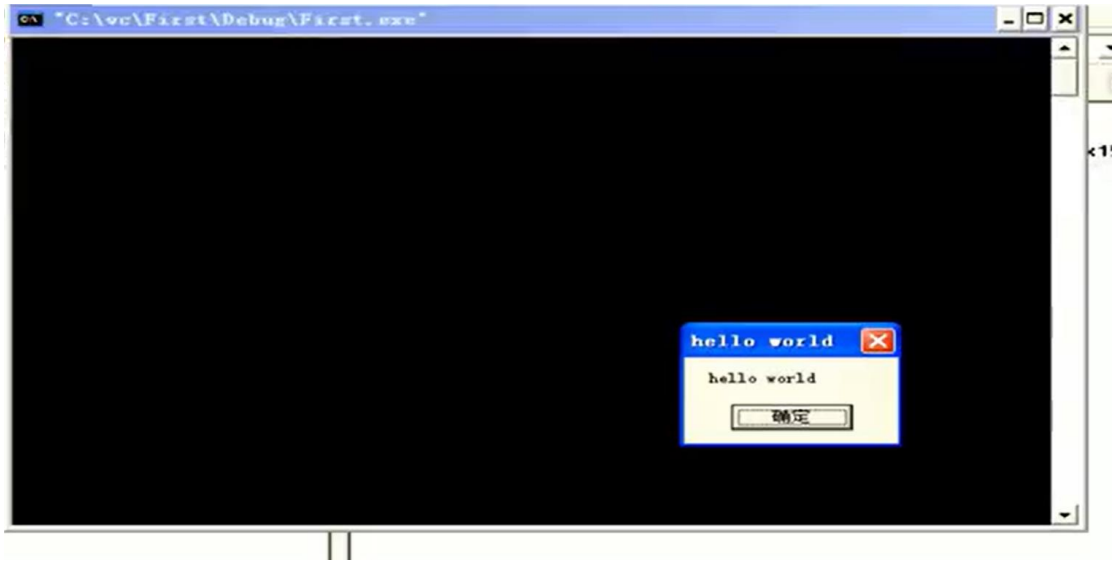
现在要解决的问题是获取 eax 的值，根据上述讲解的 call 指令思想，我们可以得到如下改进过后的解码程序，call label 使返回地址即下一条指令 pop eax 的地址压入栈中，通过 pop 指令奖盖条指令的地址赋给 eax，通过计算可得 shellcode 的起始地址。

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

int main()
{
    __asm
    {
        call lable;
     lable: pop eax;
        add eax, 0x15
        xor ecx, ecx
    decode_loop:
        mov bl, [eax + ecx]
        xor bl, 0x44
        mov [eax + ecx], bl
        inc ecx
        cmp bl, 0x90
        jne decode_loop
    }

    return 0;
}
```

我们将该解码程序的 shellcode 代码提取，与之前编码过后的 shellcode 进行拼接，得到我们最终带有解码程序 shellcode 编码。将改代码写入判断是否正常运行的万能程序：

```
#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\
void main()
{
    LoadLibrary("user32.dll");
    int *ret;
    ret=(int*)&ret+2;
    (*ret)=(int)ourshellcode;
    return;
}
```

最终成功运行 textmessage 程序：



心得体会：

通过对 shellcode 编码解码实验的复现，明白了什么是 shellcode，如何利用 shellcode 对软件进行攻击，在汇编代码的编写方面，学会了利用 call 指令获取当前指令地址，对以后汇编代码的编写与理解有很大的帮助。