《软件安全》实验报告

姓名：李雅帆　　学号：2213041　　班级：信安班

一、实验名称：

Shellcode 编写及编码实验

二、实验要求：

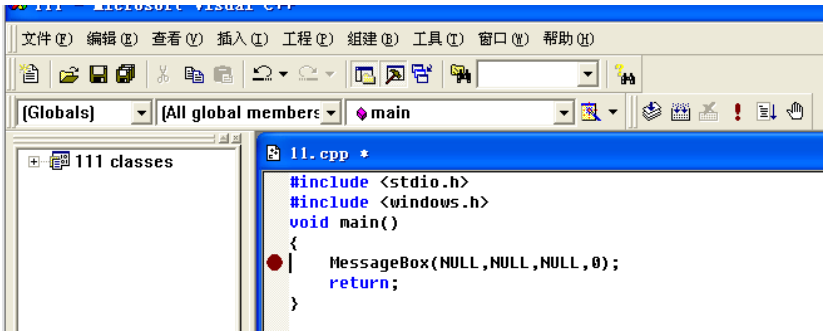
复现第五章实验三，并将产生的编码后的 shellcode 在示例 5-1 中进行验证，阐述 shellcode 编码的原理，shellcode 提取的思想。

三、实验过程：

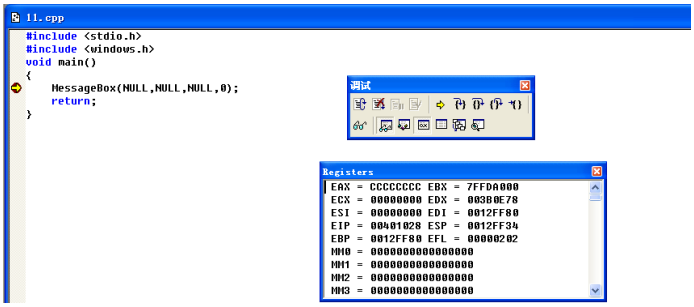
1. 提取 Shellcode 代码

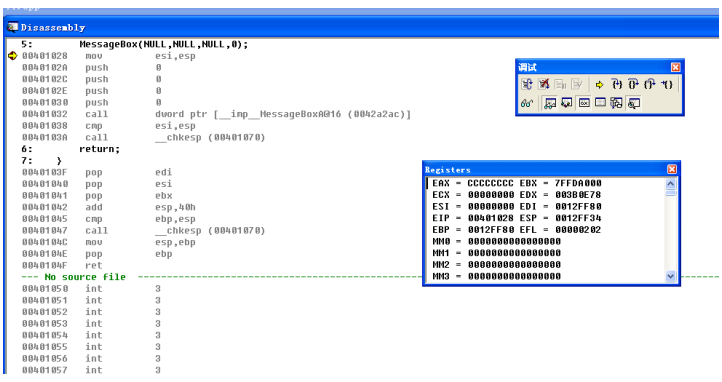
在实验中，直接写汇编代码一般难度很大，此时我们可以先采用 C++语言编写代码，来观察代码的结构，然后再按照汇编代码的结构来写汇编代码。

在 vs 中写入一段 MessageBox 的 C++代码并在固定位置打下断点



进入 vc 反汇编，可以看到代码的逻辑。



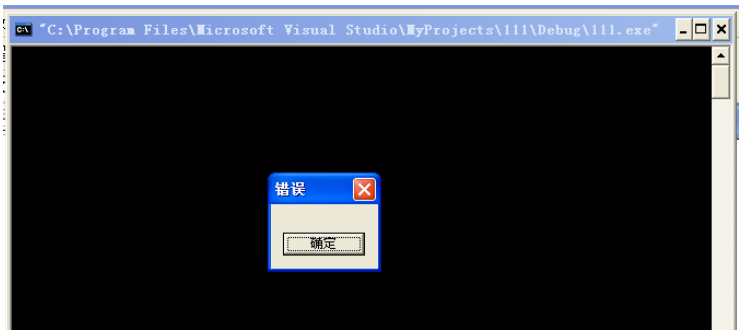


根据反汇编的代码逻辑，编写自己的汇编程序。

```
11.cpp *
#include <stdio.h>
#include <windows.h>
void main()
{
    MessageBox(NULL,NULL,NULL,0);

    _asm{
    xor ebx,ebx
    push ebx//push 0
    push ebx
    push ebx
    push ebx
    mov eax, 77d507eah
    call eax

    }

    return;
}
```

在这里，首先 xor 产生一个 0，然后 push 四个 0，这段代码无论是否删除 MessageBox 函数，编译都通过。

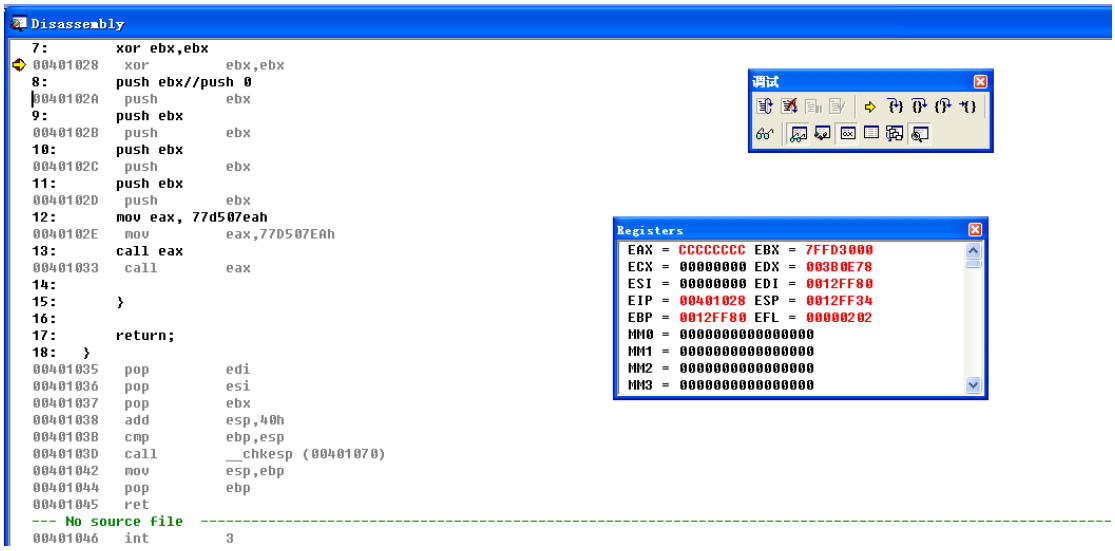
运行代码，由于在程序中我们没有提供任何字符串，所以也没有打印出来，显示错误。



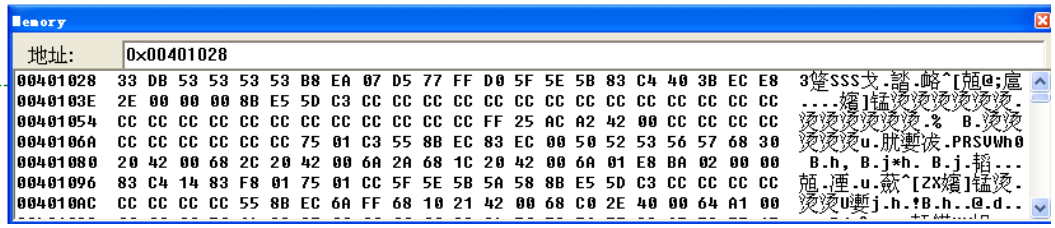
此时如果我们想得到对应程序的机器码，先在程序中加入断点。

```
11.cpp *
#include <stdio.h>
#include <windows.h>
void main()
{

    _asm{
    xor ebx,ebx
    push ebx//push 0
    push ebx
    push ebx
    push ebx
    mov eax, 77d507eah
    call eax

    }

    return;
}
```

进入反汇编模式

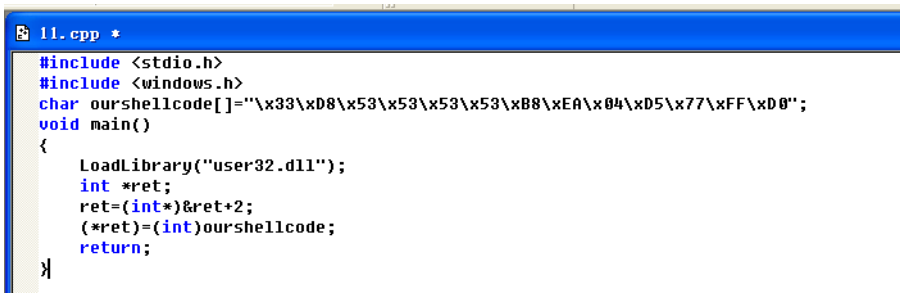


此时我们的目标不是要看它的反汇编，而是要看它的地址，此时他的地址是 00401028，我们定位到这个地址。

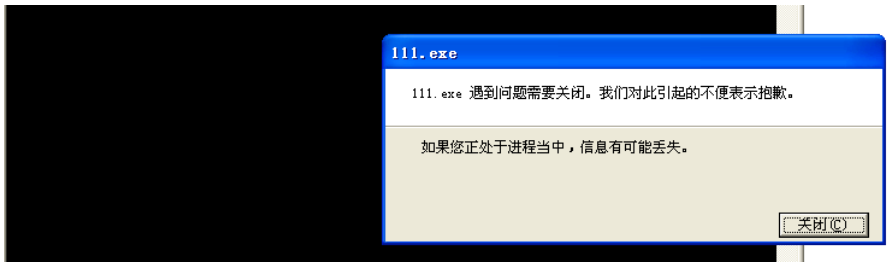


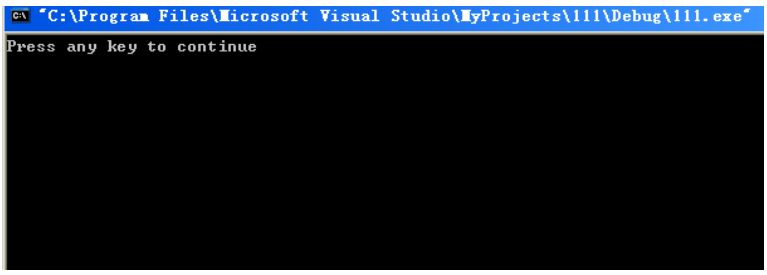
在地址为 00401035 时结束，所以在地址栏里，第一行从 33 到 D0 就是我们要找的。

此时我们新建一个文本文件并将找到的机器码存入其中，对于这个机器码，我们可以新建一个 C++程序对他进行验证和处理。



此时运行程序





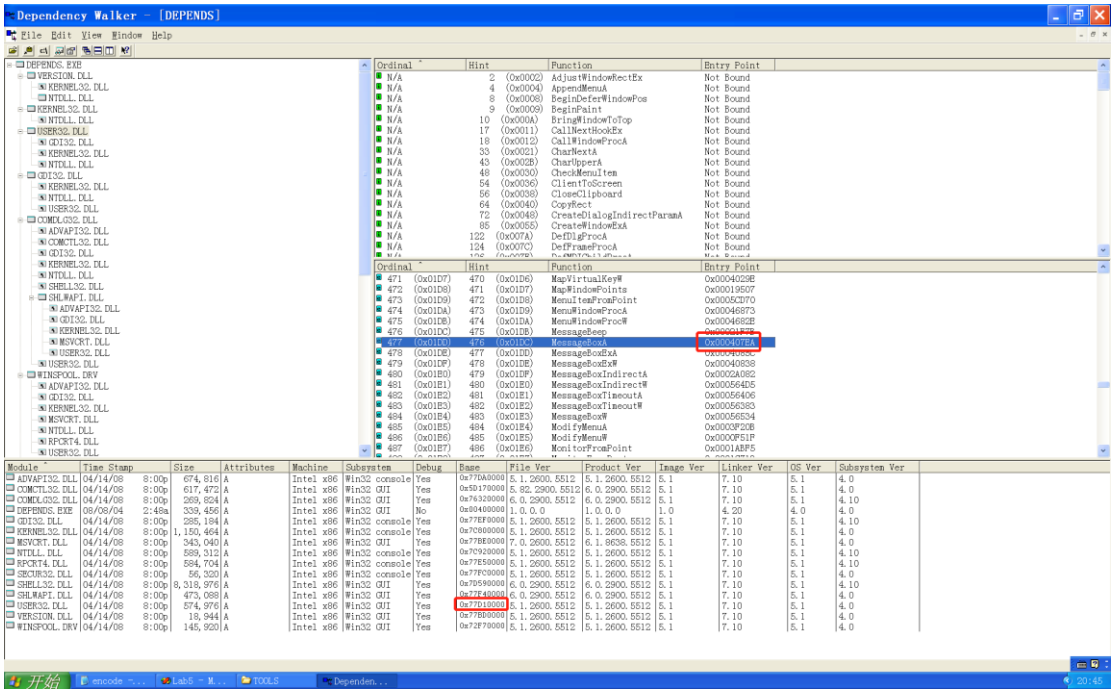
说明我们写的 shellcode 可以调用 MessageBox.

2. Shellcode 编码

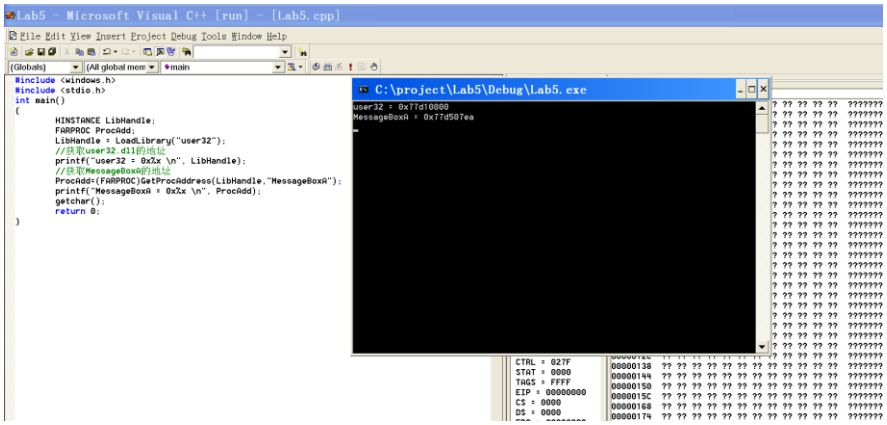
实验要求我们将一个能够弹出”Hello world”的 shellcode 进行编码，然后再讲编码后的 shellcode 和一个解码的 shellcode 合并成最终的 shellcode，实现能够将 shellcode 通过编码进行提高漏洞利用的稳定性。

我们有两种方法可以获取到 MessageBoxA 的函数入口地址，一个是通过工具包获取 user32.dll 加载的基地址以及 MessageBoxA 方法的偏移来确定函数最终的地址。

效果如下图所示，得到的 MessageBoxA 方法在内存中的地址为 0x77D507EA



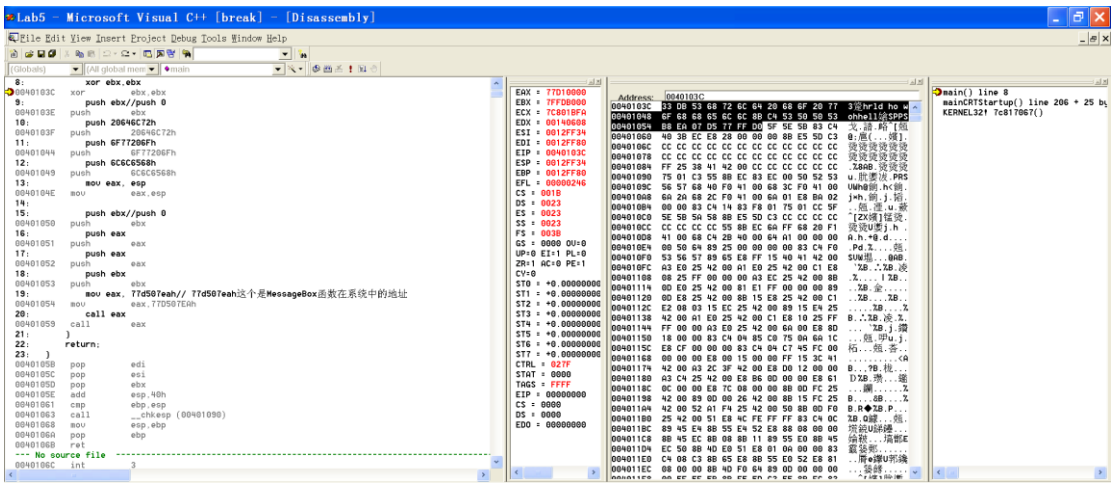
也可以通过程序来动态获取，效果如下图所示



得到了 MessageBoxA 的入口地址之后，我们就可以首先将函数所需要的 4 个参数依次入栈，然后通过 CALL 指令直接调用函数。

```
int MessageBox(
    HWND hWnd,            // handle to owner window
    LPCTSTR lpText,       // text in message box
    LPCTSTR lpCaption,    // message box title
    UINT uType            // message box style
);
```

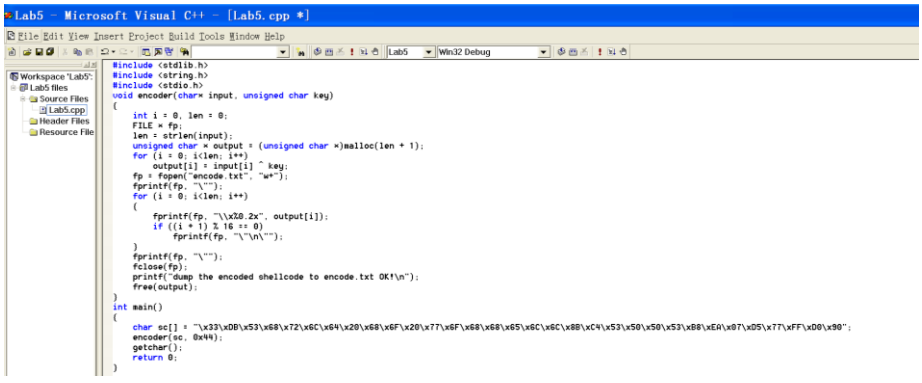
为了能够在弹出的消息窗口中展示出”Hello World”，我们需要将”Hello World”的字符串首先存入到内存中，我们直接将字符串压入到栈中，并在考虑到存储时的大端效应，因此我们要将字符串倒叙压入栈中，并且在最后补上空格凑足字节，而且首先压入了一个 0，保证字符串有串尾’\0’。



通过加入断点，查看反汇编代码，然后根据汇编代码的地址直接定位机器码的内容，得到了能够弹出”Hello world”消息框的 shellcode，得到 shellcode 如下：

\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

然后编写对于 shellcode 的编码程序，并将编码后的结果输出到文件中。编码程序采用的是将得到的机器码和指定的一个 key 值进行异或操作，原因是异或操作是可逆的，程序代码如下：



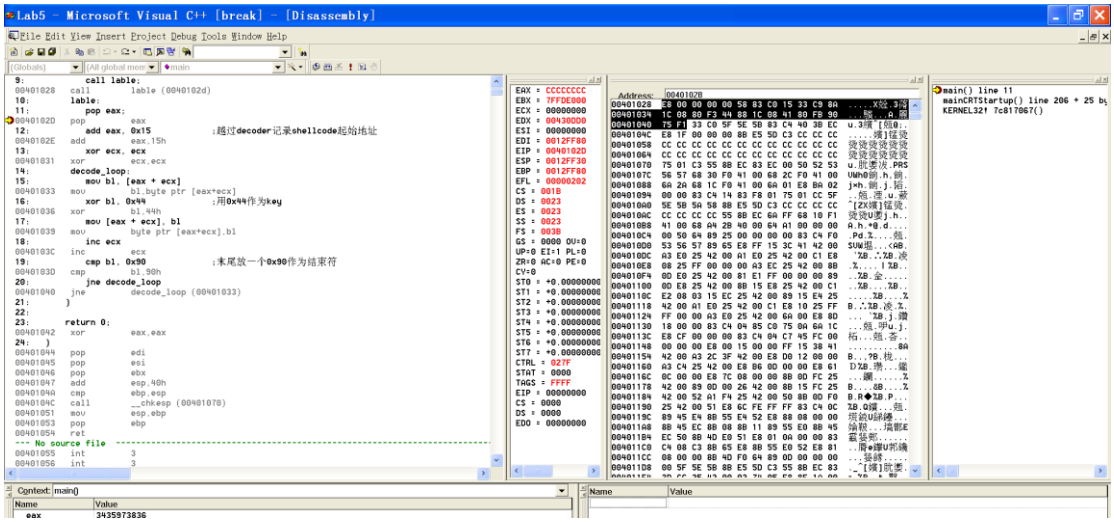

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
void encoder(char* input, unsigned char key)
{
    int i = 0, len = 0;
    FILE * fp;
    len = strlen(input);
    unsigned char * output = (unsigned char *)malloc(len + 1);
    for (i = 0; i<len; i++)
        output[i] = input[i] ^ key;
    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\x90";
    encoder(sc, 0x44);
    getchar();
    return 0;
}
```

得到编码后的 shellcode 如下：

\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

如何在 shellcode 进行漏洞攻击的时候，能够将编码好 shellcode 进行解码呢？

我们考虑可以编写一个解码程序，首先将主函数的控制权转一个解码程序，也就是将解码程序的 shellcode 放在消息框 shellcode 的前面，将函数的返回地址修改为解码程序 shellcode 的首地址，但是为了能解码消息框 shellcode，我们还需要确定消息框 shellcode 的位置，因此我们还要先确定解码程序 shellcode 的长度。



通过 CALL 指令，将下一条指令的地址压入栈中，由于 pop 就是 label 中的第一条指令，这时通过 pop 就可以取到当前指令的地址。然后根据得到的机器码的长度，确定出消息框的 shellcode 在其后面的相对偏移，而这个偏移恰好是 15h

因此我们就得到了解码程序的 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

而之前获得的弹出消息框的机器码经过编码后得到的 shellcode 如下，

\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

将两者拼接起来就可以得到我们最终的 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

4. 实验复现

尝试发现 5-1 无法复现，复现 5-4 如下：

四、心得体会：

1.阐述 shellcode 编码的思想

Shellcode 编码的思想是将原始的二进制 Shellcode 进行编码转换，以绕过防御机制或者实现特定功能。编码的目的通常是为了隐藏 Shellcode 的真实内容，使其在传输过程中不易被检测到，或者是为了解决特定的环境限制问题。

编码的过程可以采用各种算法，常见的编码技术包括简单的异或操作、Base64 编码、ROT13 编码、插入无效指令等。这些编码技术可以改变 Shellcode 的二进制表示形式，使其在传输或者存储过程中不易被网络防御设备或者安全软件检测到。

2.阐述 shellcode 提取的思想

Shellcode 提取的思想则是在目标系统上获取 Shellcode 的执行权限并执行，以实现攻击者预期的后续操作。Shellcode 的提取过程通常包括以下几个步骤：

（1）Shellcode 注入： 将 Shellcode 注入到目标系统的合适位置，常见的注入方式包括利用漏洞进行远程注入、利用本地漏洞进行本地注入、利用社会工程学手段进行用户交互式注入等。

（2）Shellcode 执行： 一旦 Shellcode 被成功注入到目标系统，攻击者就会寻找执行该 Shellcode 的方式，可能是通过改变程序的执行流程，将控制权转移到 Shellcode 所在的内存区域，或者是利用已有的系统功能来执行 Shellcode。

（3）权限提升： 在某些情况下，Shellcode 需要特权或者管理员权限才能执行其预期功能，攻击者可能会尝试提升 Shellcode 的执行权限，以绕过系统的安全控制。

（4)功能实现：一旦 Shellcode 被执行，其所包含的代码将会实现攻击者预期的功能，这可能包括但不限于远程控制、信息窃取、文件篡改、系统破坏等操作。

总的来说，Shellcode 提取的思想是通过注入和执行 Shellcode 来在目标系统上实现攻击者的目的，而编码则是为了在此过程中保证 Shellcode 的隐蔽性和免疫性。

3.我的心得体会

在复现 Shellcode 编写及编码实验的过程中，我首先学习了 Shellcode 的基本概念和原理，了解了 Shellcode 是一段直接在系统内存中执行的机器码，常用于渗透测试和恶意软件攻击中。然后，我通过学习相关的编写和编码技术，尝试编写了一段简单的 Shellcode，并对其进行了编码处理，以绕过安全防御措施。

在验证实验中，我将编写并编码后的 Shellcode 应用于示例 5-4 中进行验证。通过验证过程，我深入理解了 Shellcode 编码的原理和提取的思想。Shellcode 编码的原理是利用编码技术对 Shellcode 进行转换，以使其在传输或存储过程中不易被检测到，从而绕过安全防御机制。而 Shellcode 提取的思想是通过注入和执行 Shellcode 来在目标系统上实现攻击者的目的，通常包括注入、执行、权限提升和功能实现等步骤。

通过这个实验，我不仅加深了对 Shellcode 编写和编码的理解，还进一步认识到了安全防御的重要性，以及攻击者对于绕过防御机制的努力和手段。