《软件安全》实验报告

姓　　名	薛旗	学　　号	20155362
班　　级	软信-1503	指　导　教　师	徐剑
开　设　学　期	2017-2018 第 二 学 期		
实　验　题　目	缓冲区溢出		
实　验　日　期	2018.05.05		
评　定　成　绩		评　定　人　签　字	
		评　定　日　期	

东北大学软件学院

实验一 缓冲区溢出攻击

一、实验环境

Microsoft Windows Server 2003

Enterprise Edition

Service Pack 1

二、实验原理（简述）

通过向程序的缓冲区写超出其长度的内容，造成缓冲区溢出，从而破坏程序的堆栈，使程序转而执行其他指令，达到攻击目的。利用远程溢出，攻击者可以在没有任何系统账号的情况下获得系统的最高控制权。

ms06035 漏洞是 server 服务中的漏洞允许执行远程代码,成功的利用此漏洞,攻击者可以对被攻击主机进行远程操纵,随后可安装程序,查看、更改或删除数据,更为严重的,能够造成被攻击主机蓝屏,死机,自动重启等等,其破坏性相当严重。

ms08025 漏洞,攻击者可能会利用此漏洞危及系统的安全,并获取对该系统的控制权。攻击者成功的以普通用户登录到系统后,可以在本地进行权限提升,把普通用户的权限提升到管理员的权限。得到管理员的权限后,便可为所欲为,对系统进行各种操作。

三、实验步骤

描述实验过程，关键过程要有截图。

1.利用 ms06035 漏洞进行攻击

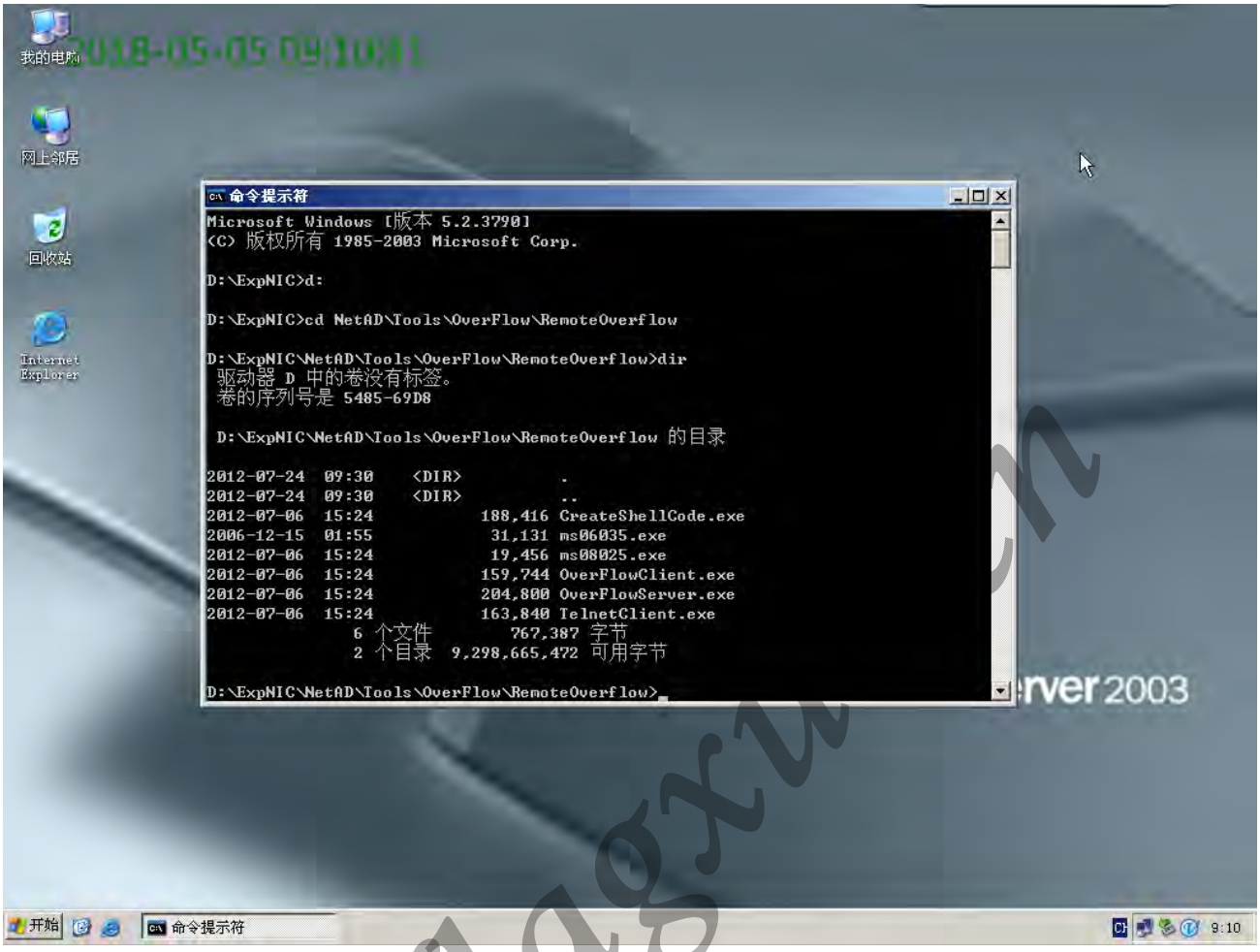
(1) 进入“ms06035 漏洞利用工具”目录。

主机 A 点击开始->运行->cmd 打开命令提示符，输入命令：

cd D:\ExpNIC\NetAD\Tools\OverFlow\RemoteOverflow 进入工作目录

(2) 查看当前目录内容。

主机 A 用 dir 命令查看当前目录中的内容，如下图所示：



上图中标注的“ms06035.exe”即为 ms06035 漏洞利用工具。

(3) 使用“ms06035 工具”进行攻击。

主机 A 执行“ms06035.exe 主机 B 的 ip 445”命令，发起对主机 B 的攻击。

(4) 主机 B 观察被攻击现象。

主机 B 被攻击后出现蓝屏死机的现象（实验结束,主机 B 点击“Hard Reboot”恢复实验环境）。

2.利用 ms08025 漏洞进行攻击

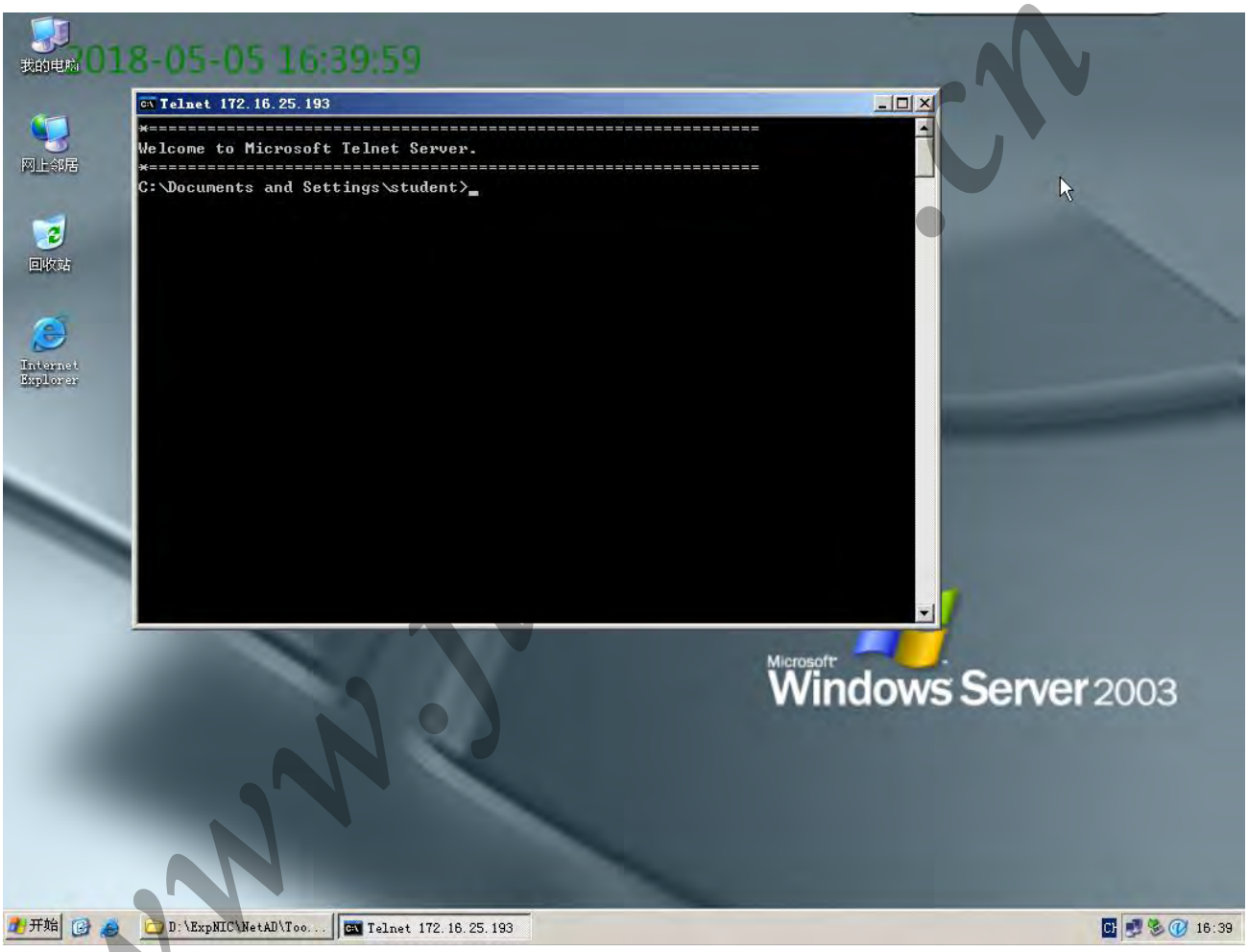
以下步骤两主机互相攻击对方，操作相同，故以主机 A 为例说明实验步骤。

「注」主机 B 进入 D:\ExpNIC\NetAD\Tools\OverFlow\RemoteOverflow 目录。将

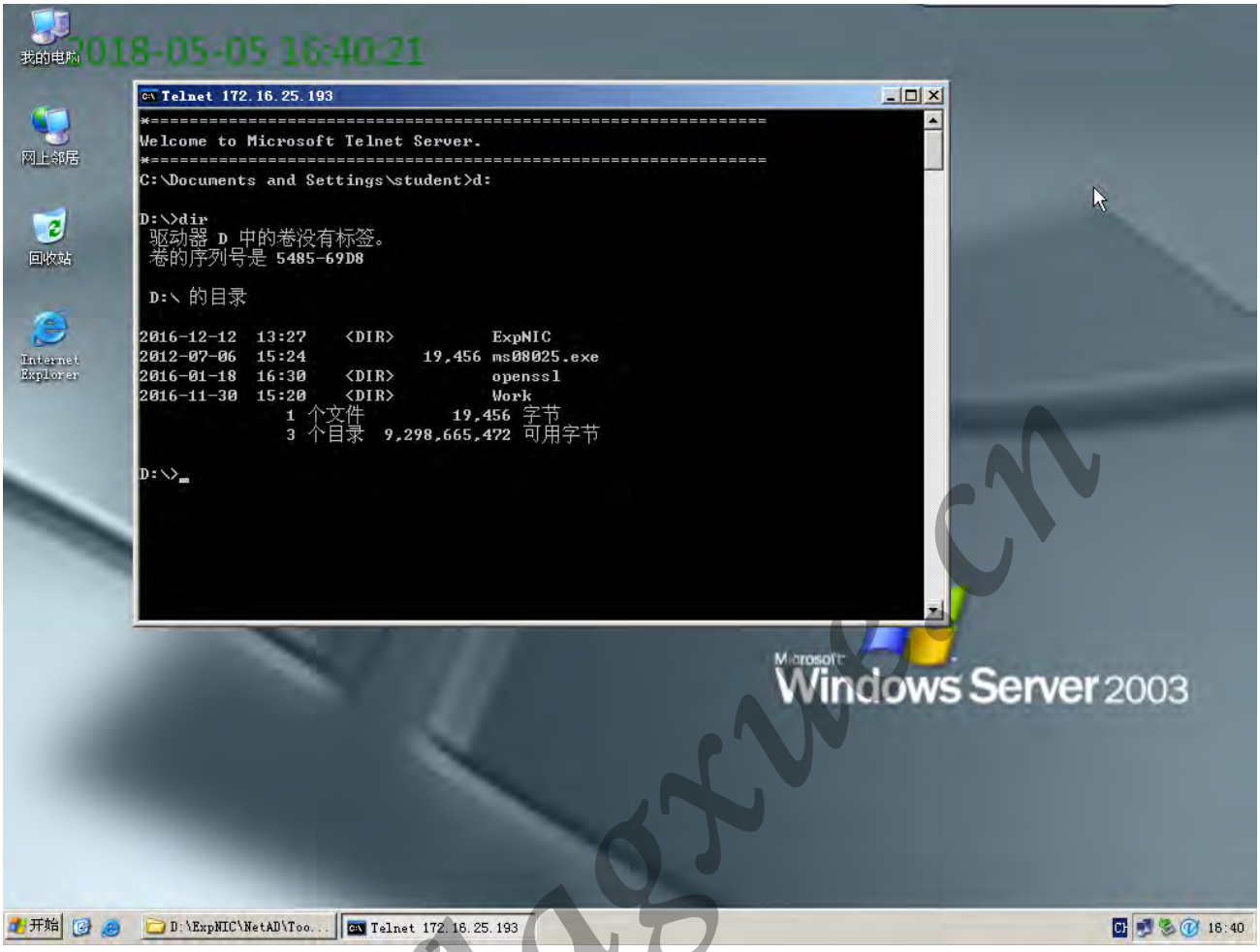
ms08025.exe 复制到 D 盘的根目录下，以便实验下一步进行。

(1) telnet 登录系统。

主机 A 在命令行下使用 telnet 登录同组主机，当出现“您将要把您的密码信息送到 Internet 区内的一台远程计算机上，这可能不安全，您还要发送吗（y/n）”当出现此提示时，选择 n，输入登录账号“student”，密码“123456”。登录成功如下图所示。

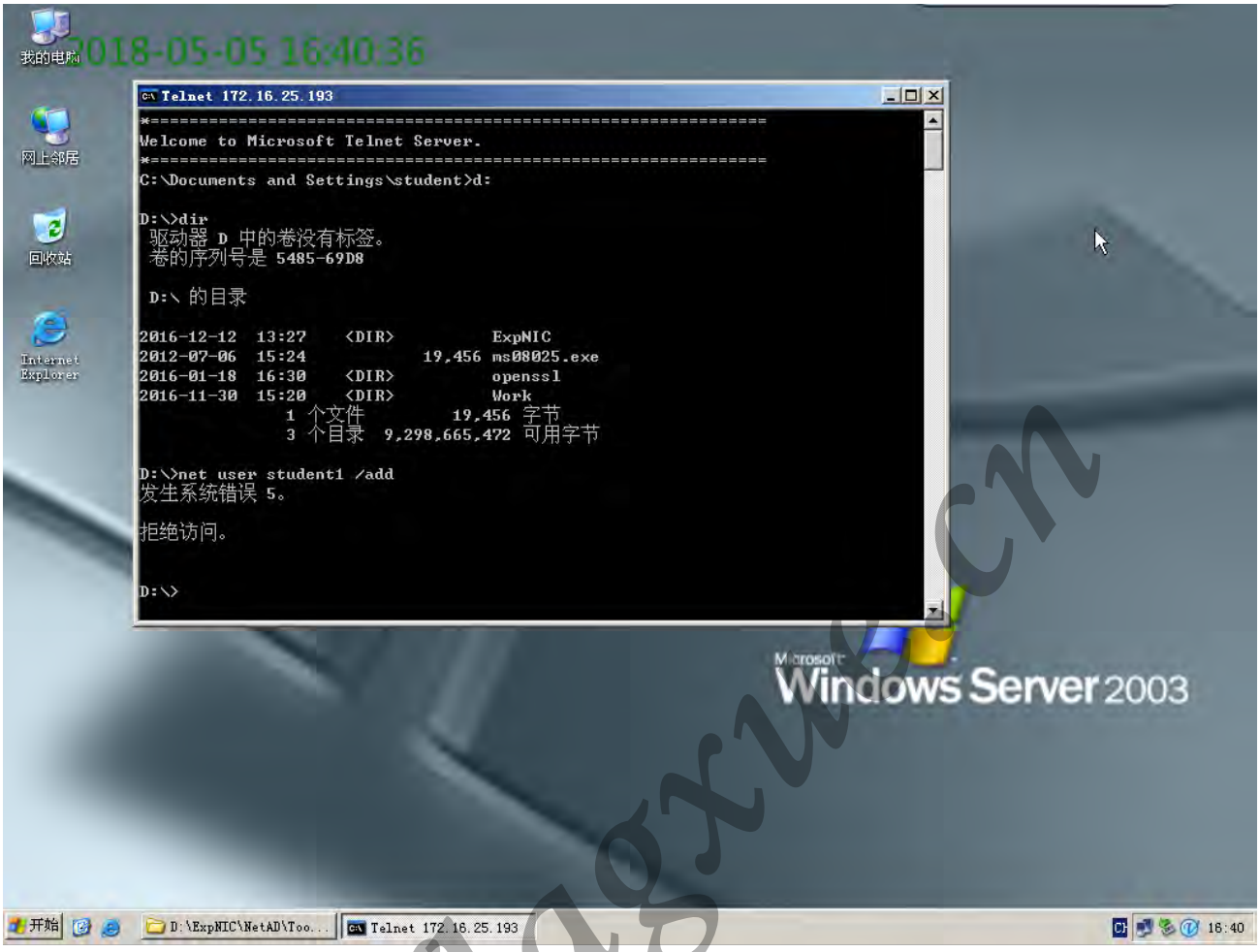


主机 A 依次输入“d:”|“dir”查看同组主机 D 盘根目录，“ms08025.exe”即为实验工具。



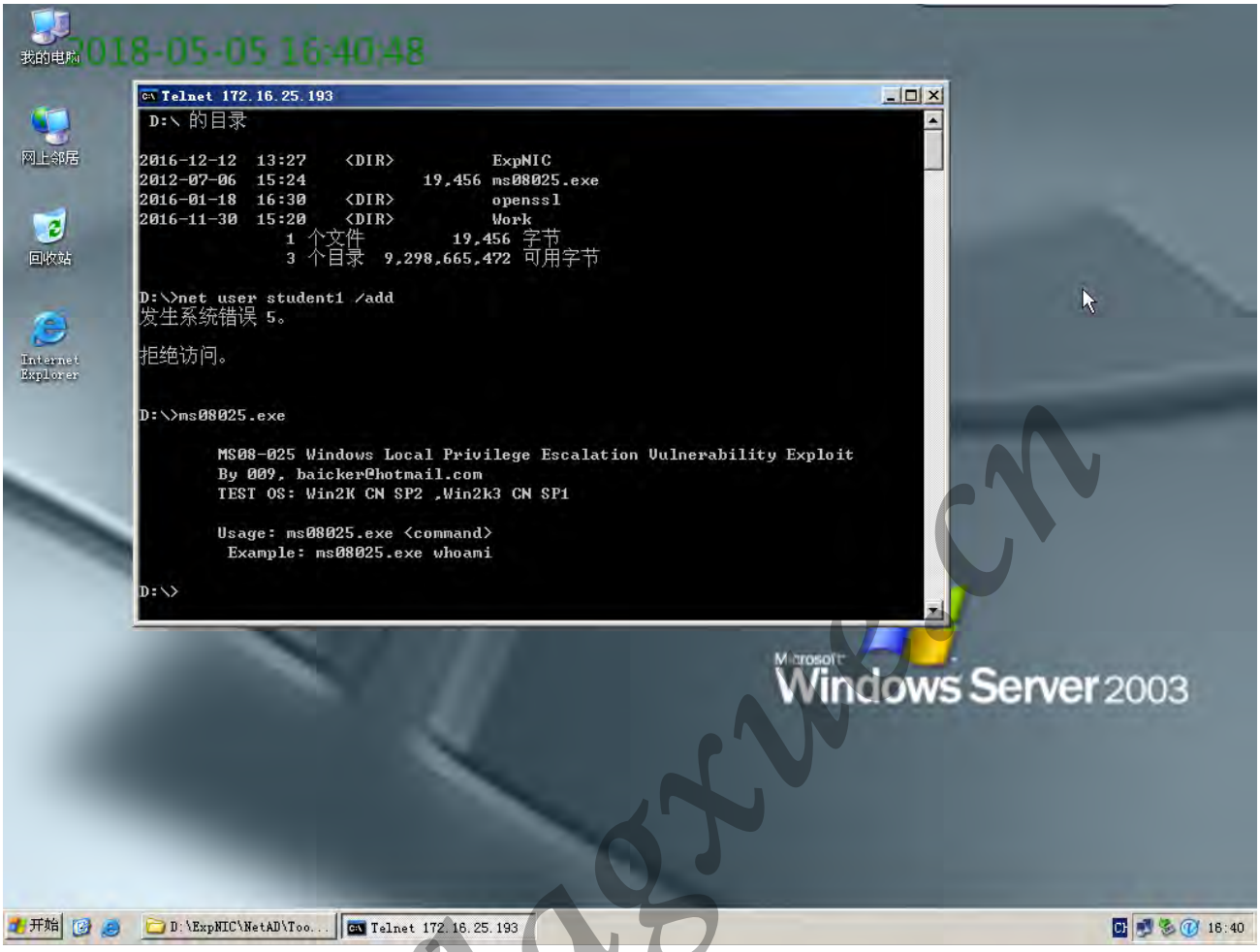
(2) 使用系统命令添加用户。

主机 A 使用“net user student1 /add”命令来试添加一个用户“student1”，执行该命令，出现“发生系统错误 5，拒绝访问”的提示，如下图所示：



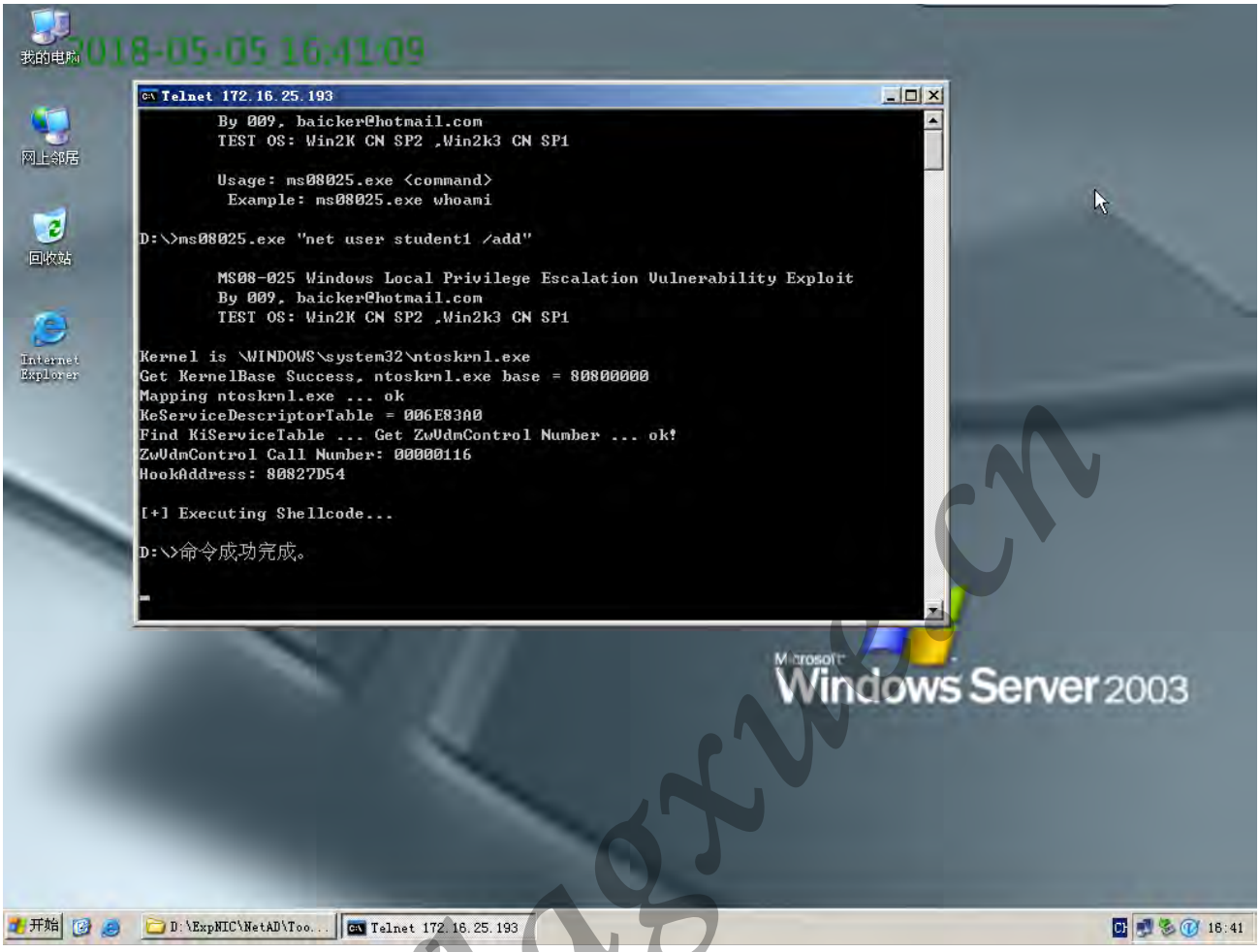
(3) 查看 ms08025 工具使用方法。

主机 A 在 telnet 命令行中输入“ms08025.exe”，查看工具的使用方法，如下图所示



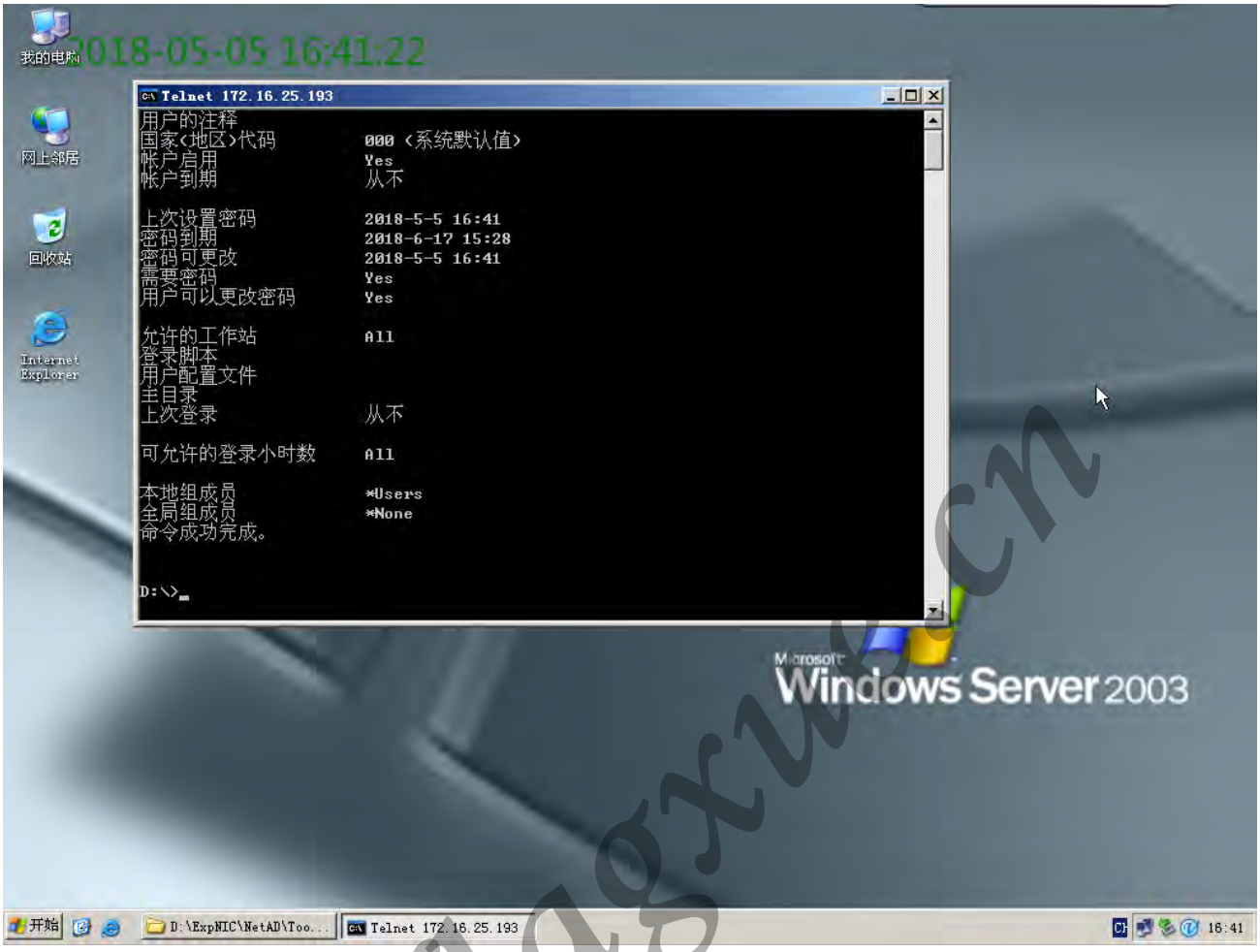
(4) 使用 ms08025 工具添加用户。

主机 A 执行“ms08025.exe "net user student1 /add"”命令，提示命令成功完成，证明用户 student1 成功添加，如下图所示：



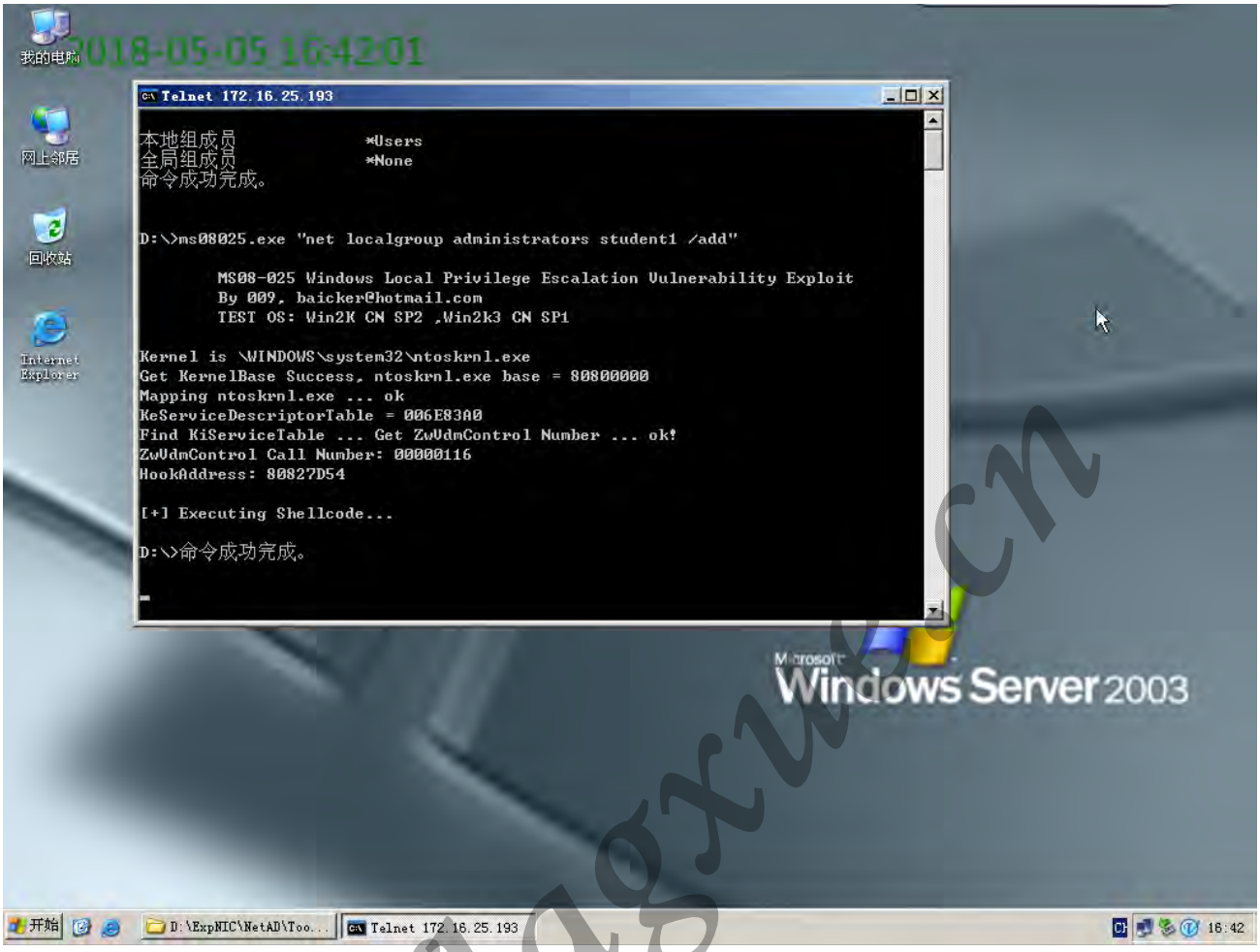
(5) 查看用户信息。

主机 A 用命令“net user student1”查看用户 student1 的信息，发现用户 student1 创建成功，隶属于 Users 组。如下图所示：

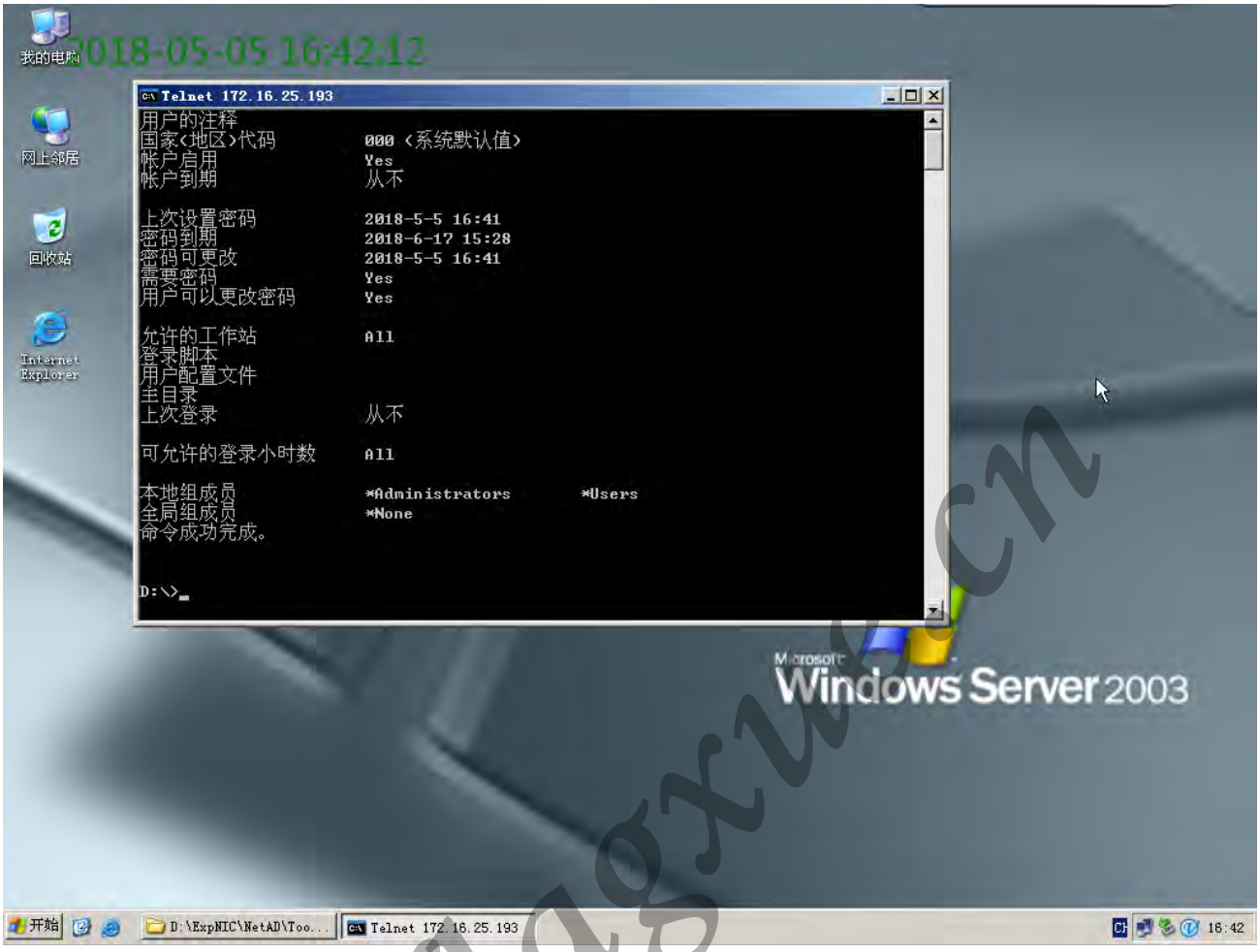


(6) 用 ms08025 工具对新建账户提权。

主机 A 执行“ms08025.exe "net localgroup administrators student1 /add"”命令将新建账户“student1”添加至管理员用户组，如下图所示：



主机A使用命令“net user student1”查看用户student1信息，可发现用户student1已被提升到管理员权限，如下图所示：



查看 student1 的用户信息，并截图。

四、实验结果总结

通过本次实验，我对缓冲区漏洞有了一个比较清晰的认识，知道了如何通过相关漏洞利用缓冲区溢出获得系统的最高控制权，破坏操作系统，知道了其危害。这提醒我在编写程序时要有责任和义务养成安全编程的思想，并熟悉那些可能会产生缓冲区溢出漏洞的函数，要对缓冲区边界进行检查。本次实验还提高了自己的团队协作能力，获益匪浅。

实验二 利用跳转指令实现缓冲区溢出

一、实验环境

Microsoft Windows Server 2003

Enterprise Edition

Service Pack 1

二、实验原理（简述）

通过向程序的缓冲区(堆、栈等)中写入超出其长度的数据,造成缓冲区溢出。缓冲区的溢出可以破坏程序执行流程,使程序转向执行其它指令。利用缓冲区溢出可以达到攻击主机的目的。

攻击者可以利用缓冲区溢出漏洞,通过溢出来获取程序的控制权。若此程序具有足够的权限,则攻击者就因此获得了系统的控制权。要实施一次有效的缓冲区溢出攻击,攻击者必须完成如下任务:

①在程序的地址空间里植入适当的代码(称为 shellcode)用于完成获取系统控制权等非法任务。

②通过修改寄存器或内存,让程序执行流跳转到攻击者植入的 shellcode 地址空间执行。

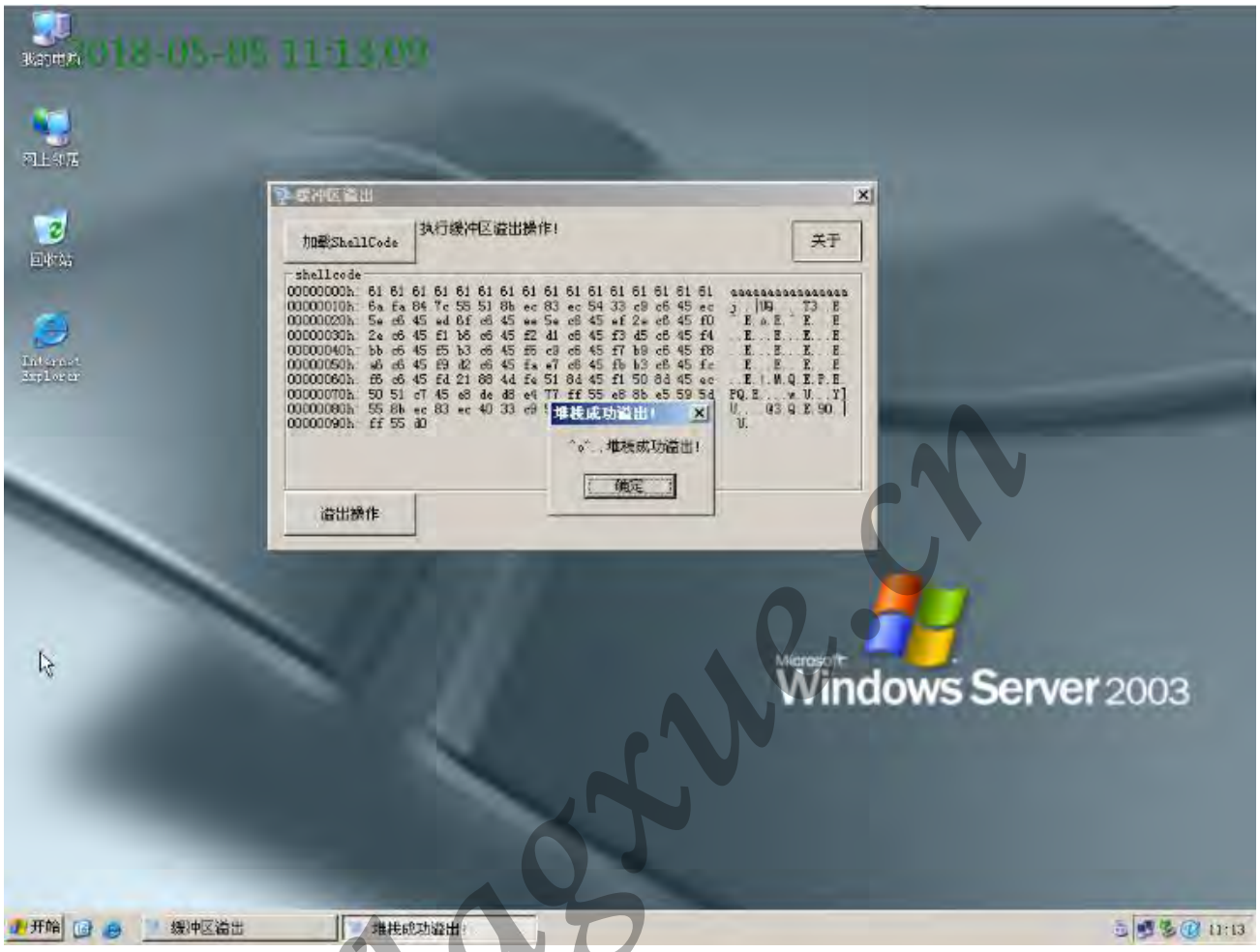
三、实验步骤

描述实验过程，关键过程要有截图。

1、溢出程序演示

点击“Vstart”工具集->网络攻防->OverFlow（D:\ExpNIC\NetAD\Tools\OverFlow）进入实验目录。进入 Mission1 目录,双击 overflow_win.exe,加载 ShellCode 执行溢出操作。

将结果截图并上传：



2、溢出实现

本练习操作通过缓冲区溢出来实现弹出消息框（MessageBox 对话框）。针对 windows 平台实现缓冲区溢出，该实验实现溢出的方式及流程具有着一定的通用性。

我们需要开发实现两部分内容：一部分是漏洞程序 overflow，该程序通过 memcpy 函数实现缓冲区溢出，当然你也可以通过其它函数实现溢出。另一部分内容则是生成 shellcode，shellcode 是程序溢出后欲执行的指令代码，如通过 shellcode 实现程序溢出后弹出对话框等功能。

在程序正常执行时，memcpy 函数被执行完毕后，指令指针会返回至 ret 地址处，继续执行 memcpy 函数调用处的后续指令；同时，执行完 ret 指令后 ESP 指针也会指向堆栈原始区（调用 memcpy 函数前一时刻的堆栈分布）。因此，我们可以将溢出代码 shellcode 存在堆栈原始区，而剩下的工作就是在 memcpy 执行返回时让 EIP 指针指向原始区(也就是 ESP 指针指向的地址)即可。如何通过 ret 返回地址确定此时的堆栈 ESP 指针指向呢？在这里采用的方法是通过跳转指令“jmpesp”(无条件跳转至 esp 指向处执行)。通过在用户地址空间中查找到

包含有“jmpesp”指令的存储地址，用该地址覆盖 ret 返回地址就可以了。

在具体实现时，我们通过三个步骤完成缓冲区溢出：

①编写前导码。

所谓前导码就是用于覆盖局部变量到ret返回地址之间的堆栈空间(不包括ret返回地址空间)的指令码。前导码仅是用于填充堆栈，所以其内容不受限制。我们需要在实际的调试中来确定前导码的大小。

「说明」cl、gcc 等诸多 C 编译器在为局部变量申请内存空间时，经常要多出若干字节。

②查找 jmpesp 指令地址。

用”jmpesp”指令的地址覆盖 ret，就可以在 memcpy 执行返回后，让 CPU 执行跳转指令，所以首要解决的是在用户空间中找到含有“jmpesp”指令的地址。通过 VC++6.0 的反汇编功能得到“jmpesp”指令的机器码为 0xFFE4。利用 FindJmpesp 工具进行指令查找，确定一个含有“jmpesp”指令的内存地址。

「注」在用户地址空间中会存在多个包含有 jmpesp 指令的地址。

③shellcode 功能体。

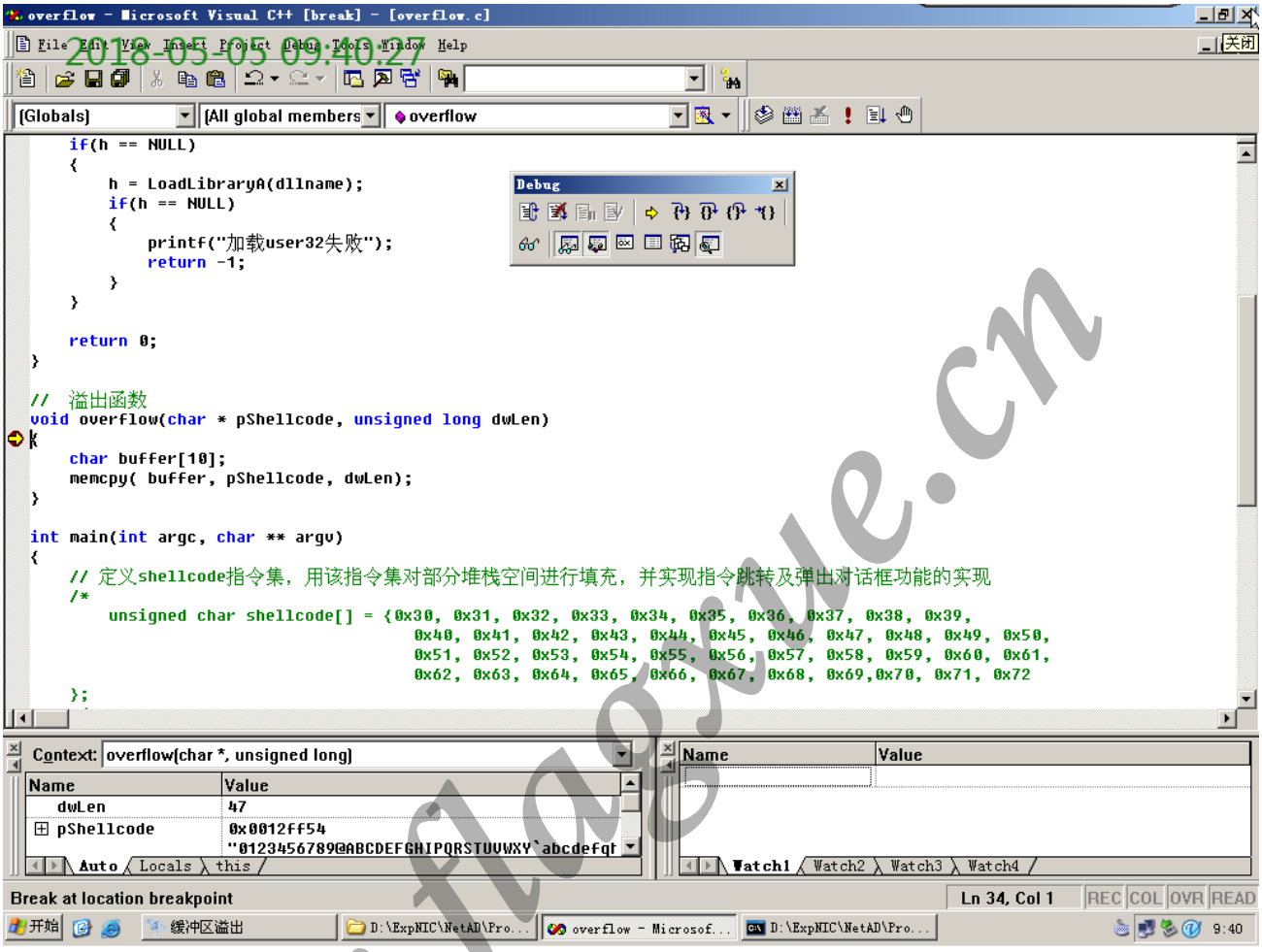
shellcode 功能体实现了溢出后主要的执行功能，如创建超级用户，提升用户权限等。在这里我们通过自定义指令来实现弹出用户对话框。

（1）编写前导码

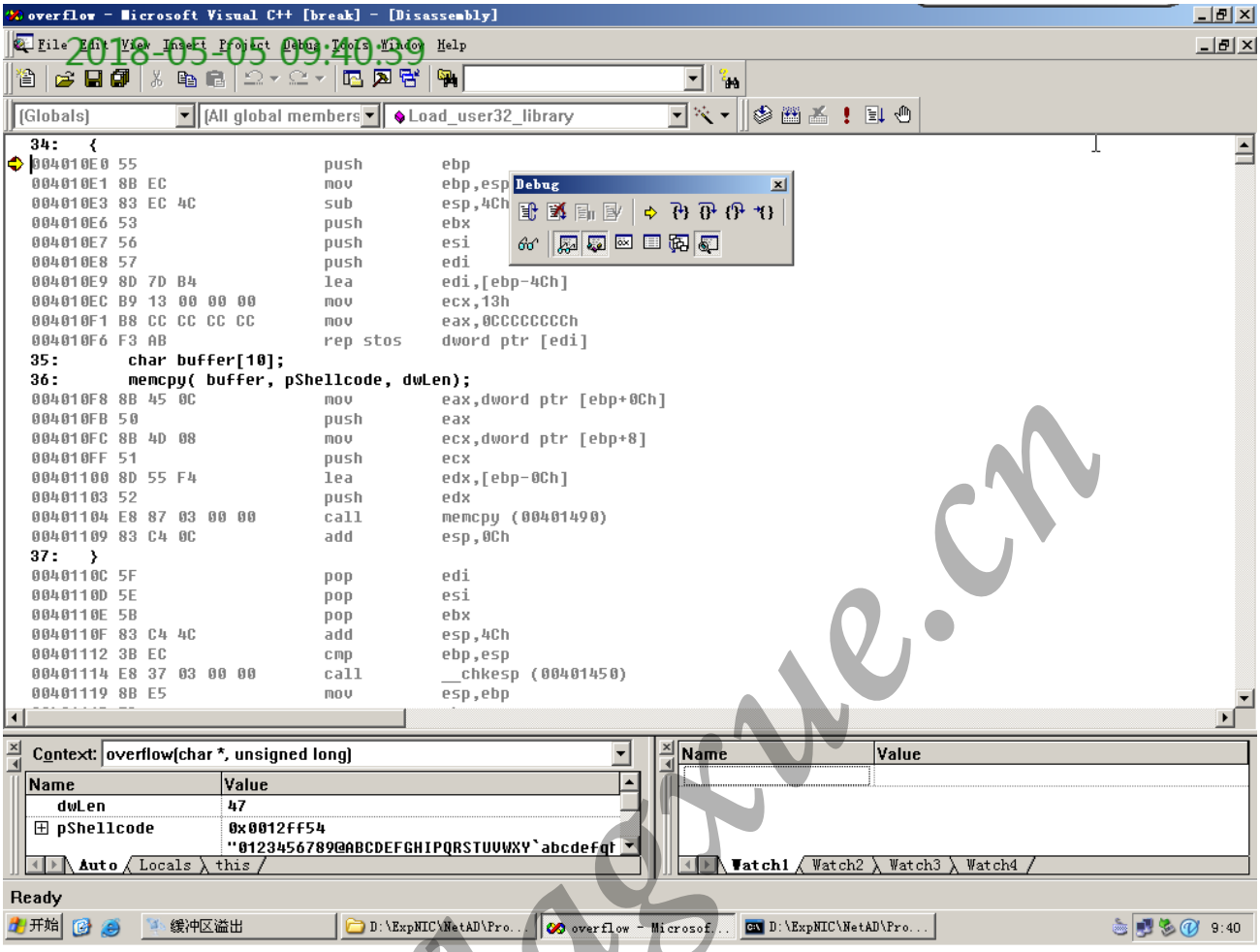
点击“Vstart”工具集->网络攻防->OverFlow2（D:\ExpNIC\NetAD\Projects\OverFlow），进入实验目录，打开工程文件。该工程包含两个项目，overflow 和 CreateShellcode 项目，建议在 debug 版下进行开发调试。

将 overflow 项目设置为启动项目（Set as Active Project），该项目仅有一个源文件 overflow.c，在此源文件中提供了部分代码，注释的地方需要你根据实际调试结果来填写。展开 overflow files 前的加号，继续展开 Source Files 前的加号，双击 overflow.c，将光标调整到源代码的第 34 行的大括弧的前面，按 F9 设置断点，点击菜单栏中的 Build|Rebuild All，把修改的内容重新保存一下。

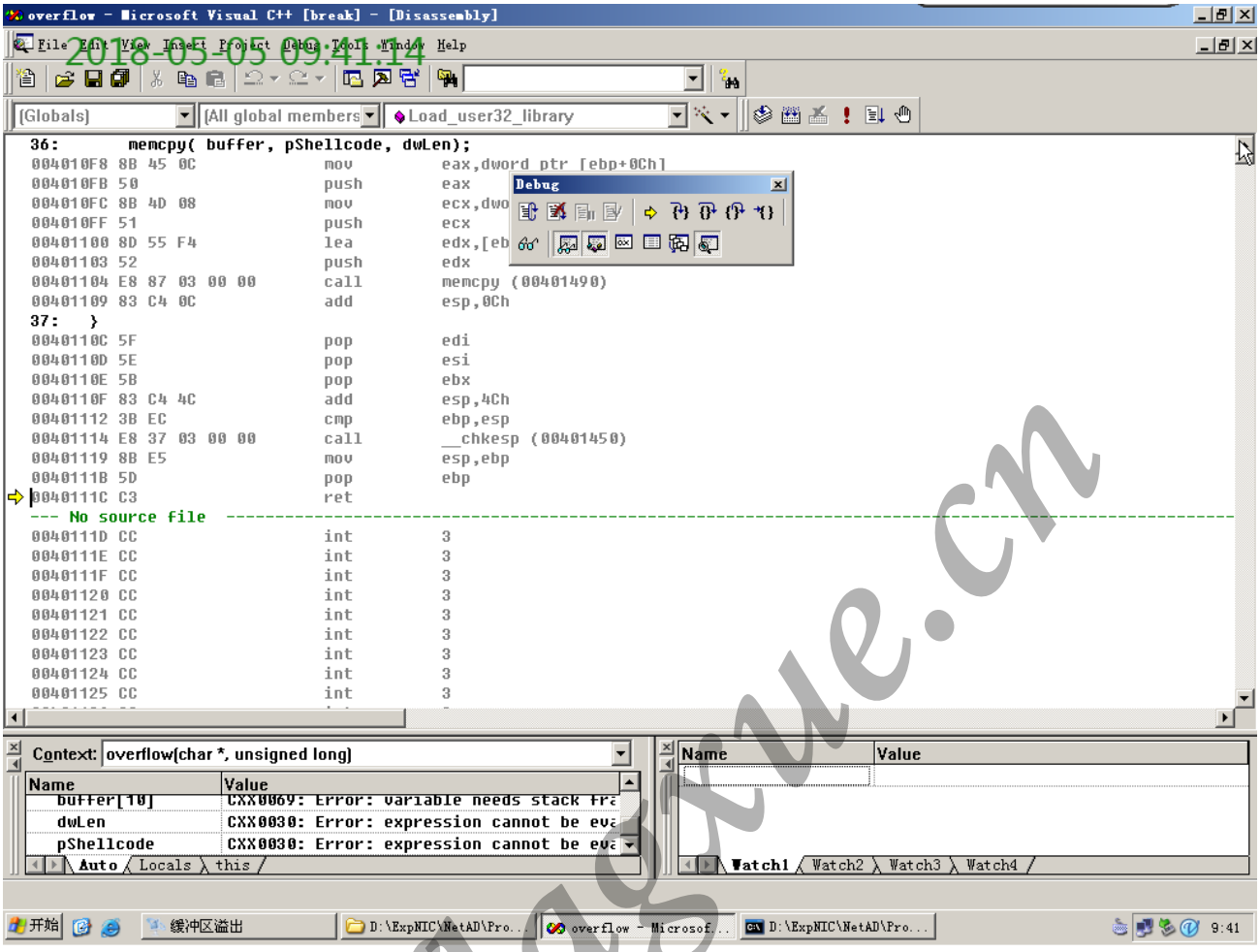
具体操作如下图：



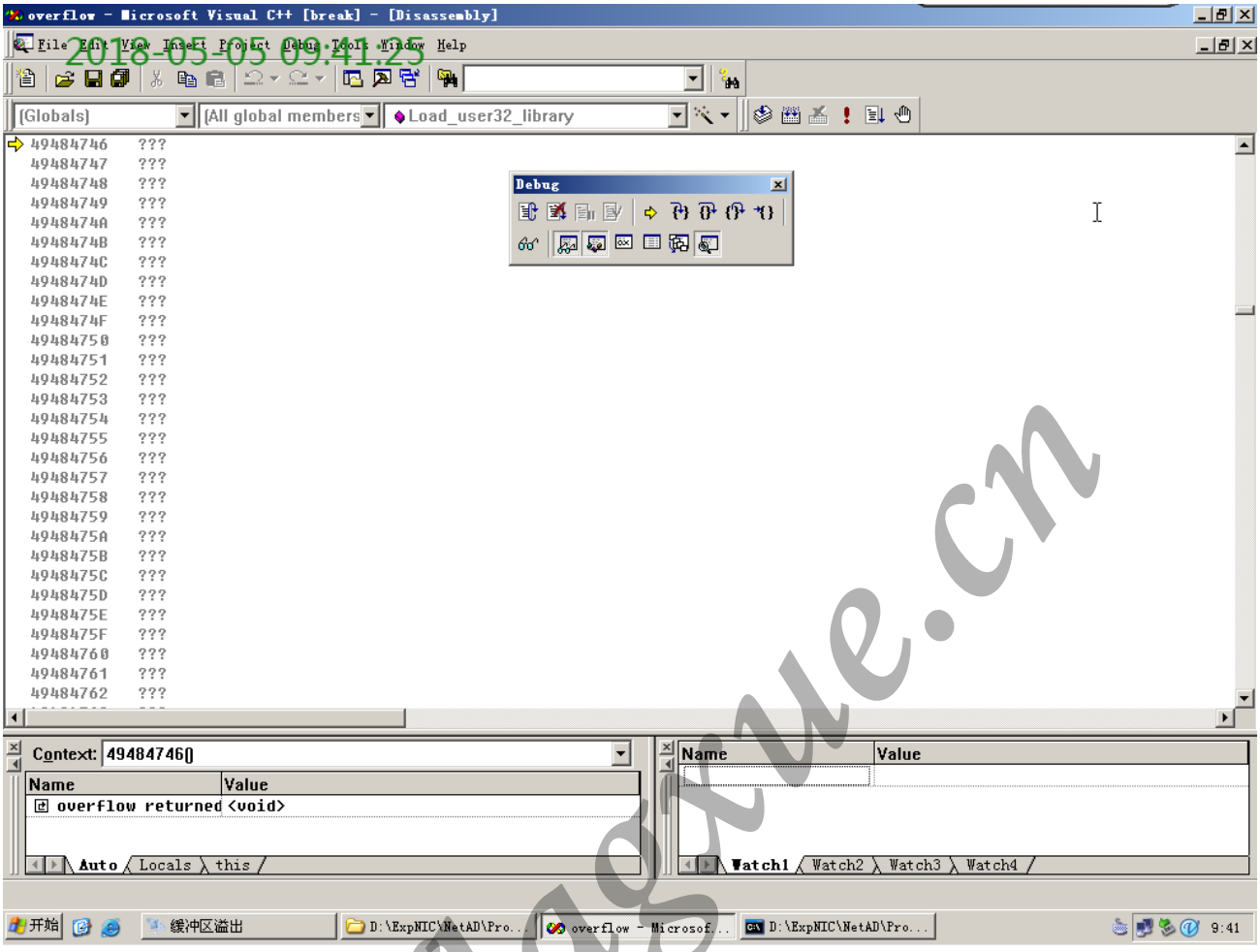
程序中提供了一个超长前导码，你需要对程序进行调试来确定实际需要的前导码长度。按 F10，对代码进行单步调试，直到调试到断点位置。按 Alt 键的同时按 8，进行反汇编，然后在代码的任意空白处点击鼠标右键，选择 Code Bytes，如下图：



然后继续按 F10 进行单步调试，将看到 overflow 调用返回，具体调试结果如下图：



再按一下 F10，将看到 ret 返回地址的信息，具体操作如下图：

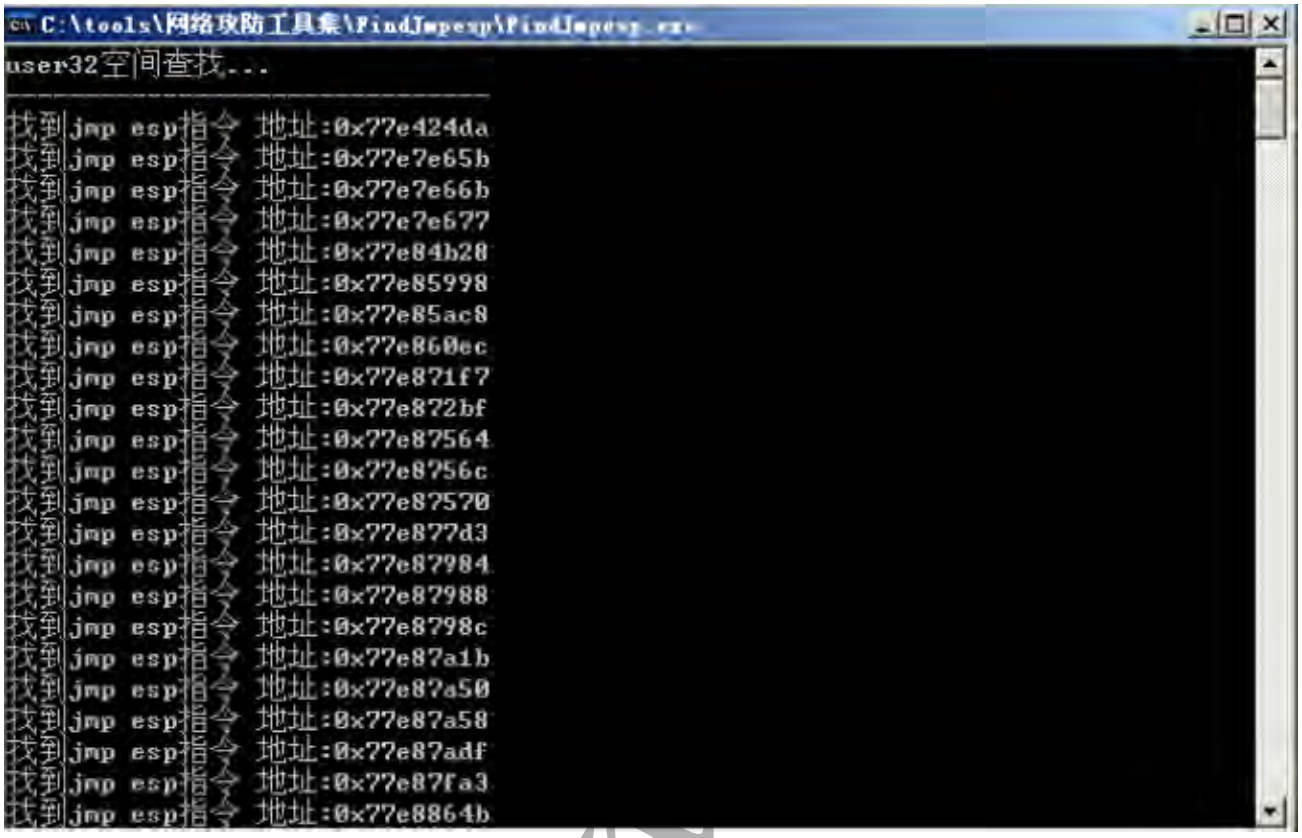


调试确定前导码长度

在上图中可以看出，0x49484746 四字节覆盖了 ret 返回地址，请根据调试结果重新确定 shellcode 指令集长度，确定 ret 返回地址能够被前导码的后续 4 字节覆盖。按 Shift 键同时按下 F5，停止单步调试，等待查找到 jmpesp 指令地址后再修改 shellcode 指令集。

（2）查找 jmpesp 指令地址

我们需要在用户地址空间中找到包含有 jmpesp 指令(机器码为 0xFFE4)的地址。运行 FindJmpesp 工具，选取一个地址追加到 shellcode 尾（追加填加地址时注意数组高字节对应地址高位），所选 jmpesp 指令地址是 0x77e424da。



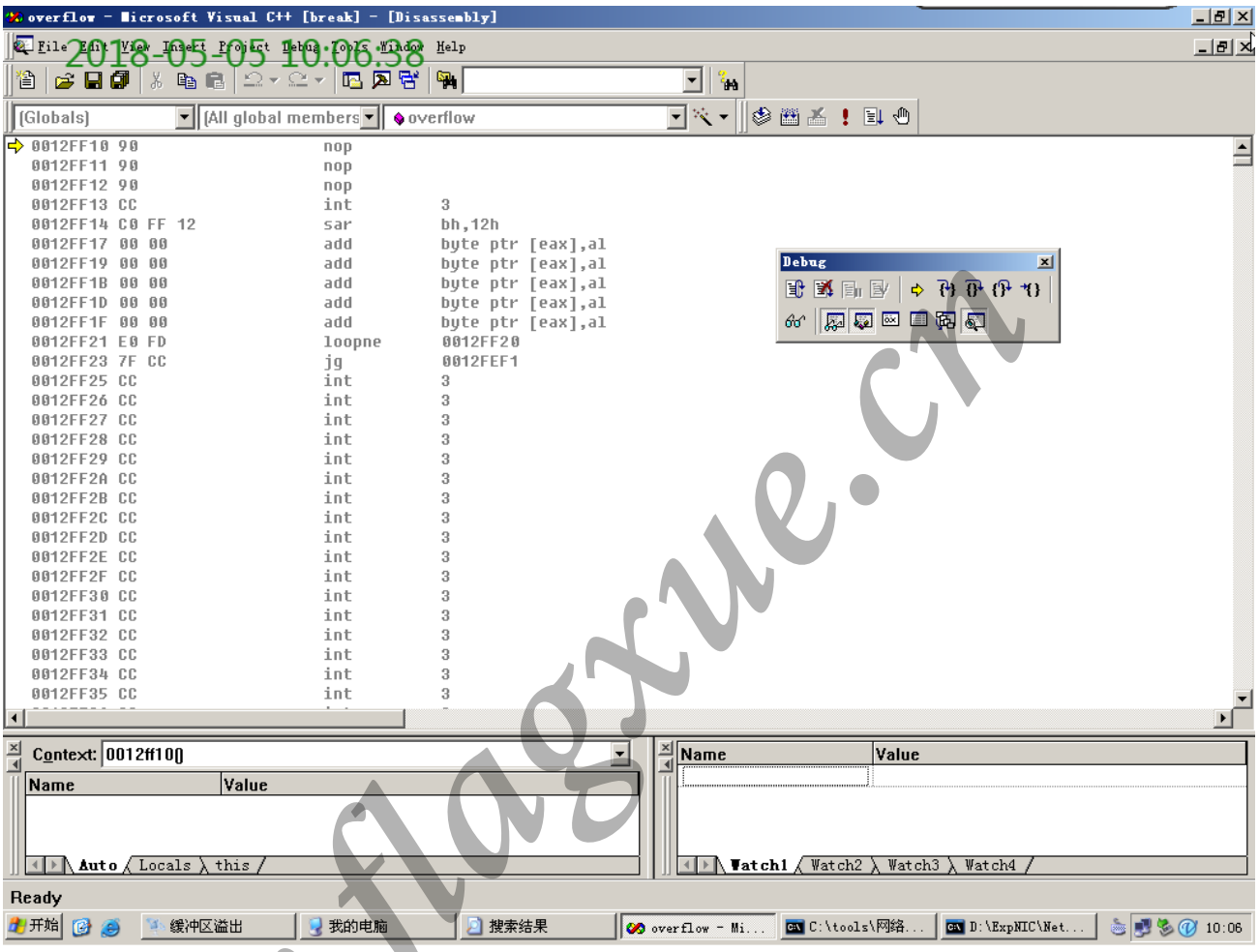
跟踪调试程序，确定在 memcpy 执行返回时 jmpesp 指令是否被执行。调试过程如图所示。

「说明」可以在 shellcode 尾部继续追加空指令(0x90，空指令不进行任何操作)，这样便于确定执行 jmpesp 后指令指针的指向。

```
unsigned char shellcode[] = {0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39,
                             0x40, 0x41, 0x42, 0x43, 0x44, 0x45,
                             0xda, 0x24, 0xe4, 0x77, 0x90, 0x90, 0x90
};
```

```
00401100 5F                   pop         edi
00401101 5E                   pop         esi
00401102 5B                   pop         ebx
00401103 83 C4 4C             add         esp,4Ch
00401106 3B EC                cmp         ebp,esp
00401108 E8 37 03 00 00       call        __chkesp (00401444)
0040110D 8B E5                mov         esp,ebp
0040110F 5D                   pop         ebp
00401110 C3                   ret
```

```
77E424DA FF E4                jmp         esp
77E424DC 03 00                add         eax,dword ptr [eax]
77E424DE 00 76 10             add         byte ptr [esi+10h],dh
77E424E1 81 FF E5 03 00 00    cmp         edi,3E5h
```



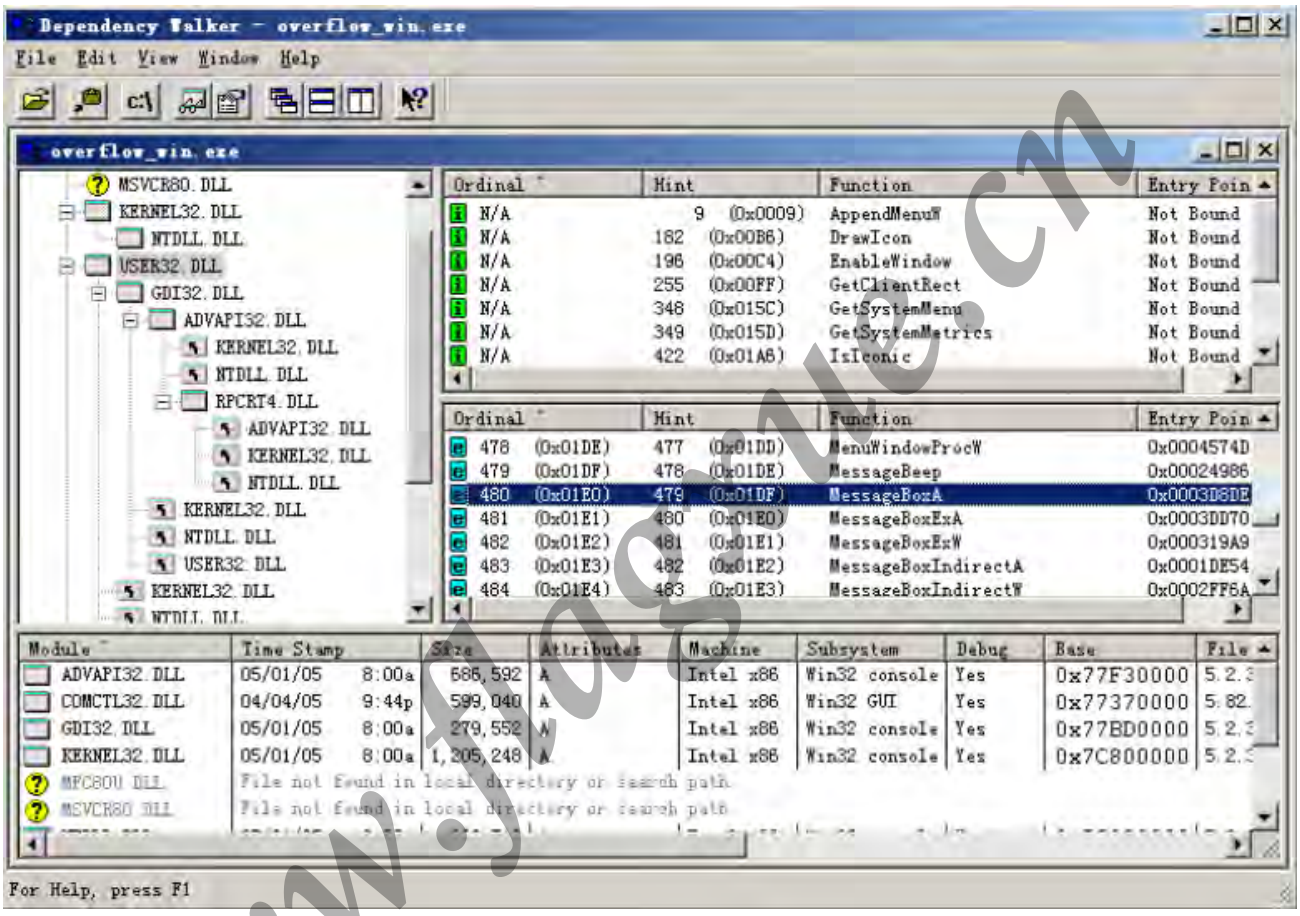
在 jmpesp 后是 4 个空指令

从上图可以看出，在 jmpesp 指令执行完毕后，指令指针紧接着执行了 3 个空指令，而空指令是追加在 shellcode 尾部的。所以我们下一步所要做的工作就是将实现弹出对话框的指令码追加至 shellcod 中 jmpesp 指令地址的后面。

（3）生成实现弹出对话框的指令码

我们最终的目的是要通过缓冲区溢出实现弹出消息对话框，而这些功能都应该在 shellcode 得以实现。通过在 shellcode 中调用 MessageBoxA API 函数，并确定好 MessageBoxA 所需的 4 个参数：窗体句柄、标题显示、内容显示和风格即可以实现弹出指定内容的对话框。

根据 Windows API 文档，MessageBoxA 依赖于 user32.lib，也就是说它位于 user32.dll 动态链接库中。单击工具栏“Depends”按钮，启动 Depends 工具，Depends 打开应用程序 D:\ExpNIC\NetAD\Tools\OverFlow\Mission1\overflow_win.exe，可以发现它将加载 user32.dll。然后寻找 MessageBoxA 函数的内存位置。具体操作如图所示。



计算 MessageBoxA 绝对内存地址

①在左侧 Module 树状视图中选中“USER32.DLL”节点；

②在右侧导出函数列表视图中遍历 Function 属性列，查找函数“MessageBoxA”（序号 480）；

③在下侧 Module 列表视图中遍历 Module 属性列，查找模块“USER32.DLL”。

在这里的 user32.dll 中，MessageBoxA（ASCII 版本）函数的偏移量（Entry Point）为 0x0003D8DE。User32.dll(Module)在内存中的起始地址(Base)为 0x77E10000。将两者相加即可得到 MessageBoxA 函数的绝对内存地址。所以我们需要在汇编代码中正确设置堆栈并调

用 MessageBoxA 函数的绝对内存地址，该地址为 0x77E4D8DE。（字母大写）

另外还需要调用执行函数 ExitProcess(位于 KERNEL32.dll 中)，其目的就是在单击弹出框“确定”按钮后程序自动退出，函数 ExitProcess 的绝对内存地址 0x7C813039 。（字母大写）

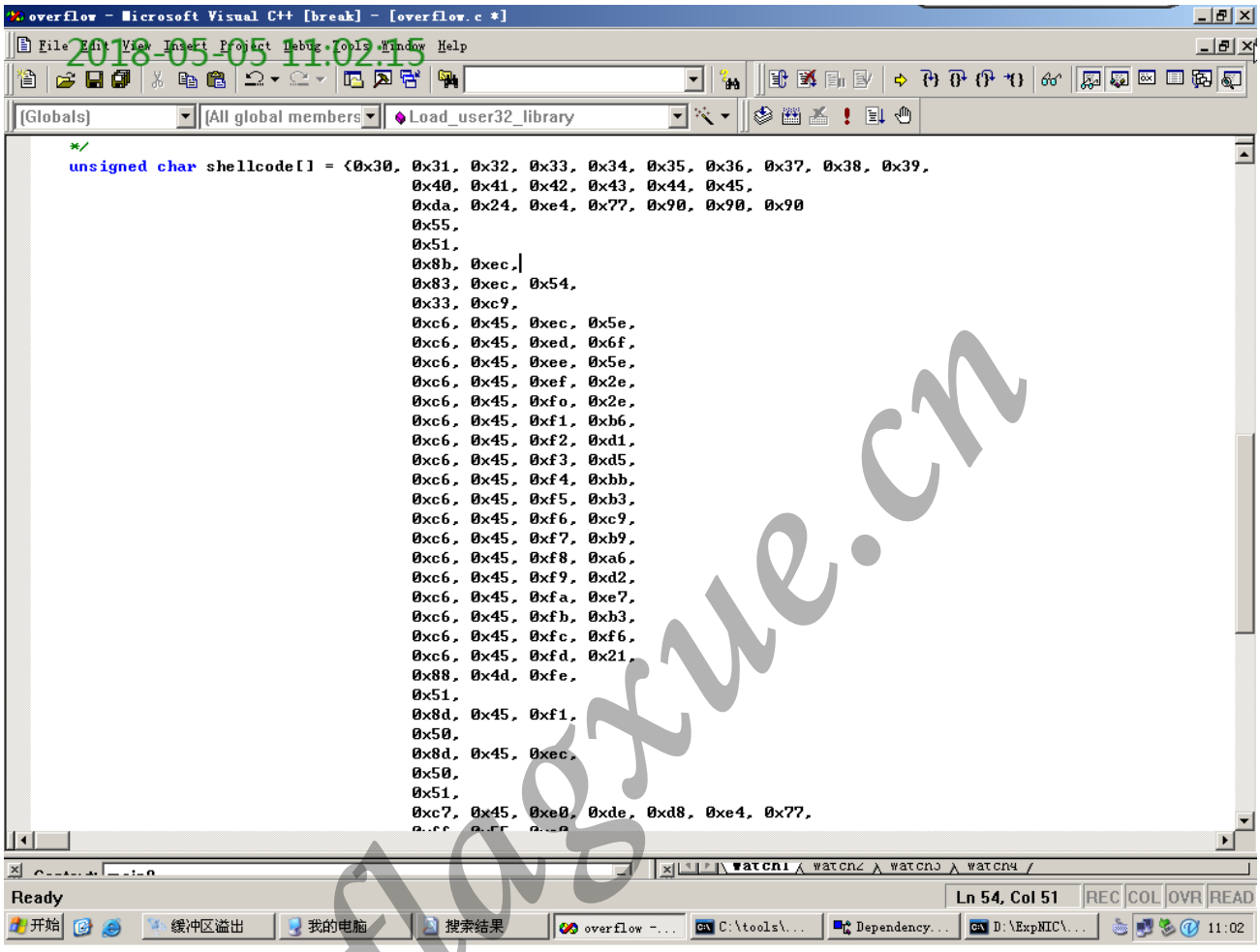
Module	Time Stamp	Size	Attributes	Machine	Subsystem	Debug	Base
ADVAPI32.DLL	05/01/05 8:00a	686,592	A	Intel x86	Win32 console	Yes	0x77F30000
COMCTL32.DLL	04/04/05 9:44p	599,040	A	Intel x86	Win32 GUI	Yes	0x77370000
GDI32.DLL	05/01/05 8:00a	279,552	A	Intel x86	Win32 console	Yes	0x77BD0000
KERNEL32.DLL	05/01/05 8:00a	1,205,248	A	Intel x86	Win32 console	Yes	0x7C800000

在 overflow 工程中将 Createshellcode 项目设置为启动项目，该项目仅有一个源文件 Createshellcode.c，在此源文件中提供了全部的代码及注释说明。代码的主体部分是用汇编语言实现的，其功能就是实现弹出对话框后自动退出程序。

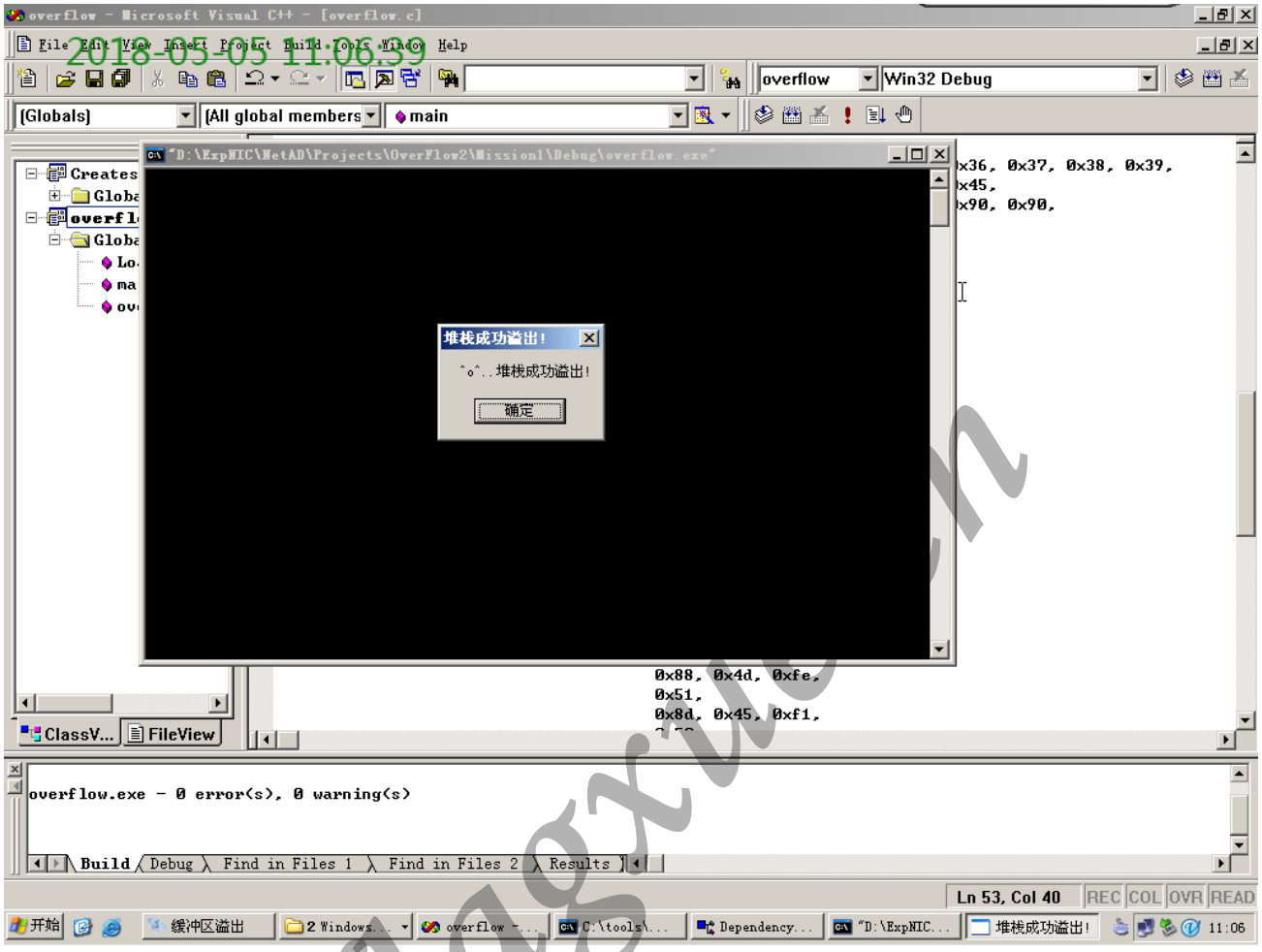
将函数 MessageBoxA 和 ExitProcess 的绝对内存地址填写到指定位置。

在理解了 Createshellcode.c 中的汇编部分代码后，就可以利用 VC++6.0 反汇编功能获取代码字节，调试过程如图所示。

显示代码字节



将代码字节以十六进制数据形式继续追加到 shellcode 尾。将运行结果截图并上传：



四、实验结果总结

通过上机实验，我对缓冲区溢出有了更加清晰的认识，知道了要完成一次有效的缓冲区溢出攻击，攻击者必须完成的两项任务：在程序的地址空间里植入适当的代码(shellcode)、通过修改寄存器或内存，让程序跳转到攻击者植入的 shellcode 地址空间执行。通过亲自实践实现了利用跳转指令实现缓冲区溢出，获益匪浅。