“ 调戏 ” 反遭 “ 反调戏 ” —记 Visual Toolbar 的破解过程

cntrump

由于要做工具栏，在网上找到了 Visual Toolbar 这个工具，看了介绍挺不错的，下载回来之后发现软件要注册，但是这是个 04 年的软件，已经没法向作者购买了，更重要的原因是俺还在失业中，吃饭都成问题了，哪里还有钱注册呀~~

用 PEID 检测是 UPX 的老壳，直接手脱，过程不是重点反正能脱壳就行了。试着注册了一下，发现我最喜爱的 MessageBox 注册错误提示，太好了，下断 bp MessageBoxA，中断下来后往上找，很容易找到一些敏感信息:

```
00407790    6A FF              push -0x1
00407792    68 60A64900        push unpacked.0049A660
00407797    64:A1 00000000     mov eax,dword ptr fs:[0]
0040779D    50                 push eax
0040779E    64:8925 0000000>mov dword ptr fs:[0],esp
004077A5    83EC 10            sub esp,0x10   ；程序用的是 MFC 所以取值部分比较啰嗦
004077A8    53                 push ebx       ;关键的代码我都用红色标出来了。
004077A9    55                 push ebp
004077AA    56                 push esi
004077AB    57                 push edi
004077AC    8BF1               mov esi,ecx
004077AE    6A 01              push 0x1
004077B0    E8 6D1B0900        call <jmp.&mfc42.#CWnd::UpdateData_6334> ;获取输入的值
004077B5    8D4C24 14          lea ecx,dword ptr ss:[esp+0x14]
004077B9    E8 40180900        call <jmp.&mfc42.#CString::CString_540>
004077BE    8D4424 14          lea eax,dword ptr ss:[esp+0x14]
004077C2    33DB               xor ebx,ebx
004077C4    50                 push eax
004077C5    8D8E 4C0E0000      lea ecx,dword ptr ds:[esi+0xE4C]
004077CB    895C24 2C          mov dword ptr ss:[esp+0x2C],ebx
004077CF    E8 AA190900        call <jmp.&mfc42.#CWnd::GetWindowTextA_3874>
004077D4    51                 push ecx
004077D5    8DBE 900E0000      lea edi,dword ptr ds:[esi+0xE90]
004077DB    8BCC               mov ecx,esp
004077DD    896424 20          mov dword ptr ss:[esp+0x20],esp
004077E1    57                 push edi
004077E2    E8 05180900        call <jmp.&mfc42.#CString::CString_535>
004077E7    51                 push ecx
004077E8    8D5424 1C          lea edx,dword ptr ss:[esp+0x1C]
004077EC    8BCC               mov ecx,esp
004077EE    896424 20          mov dword ptr ss:[esp+0x20],esp
004077F2    52                 push edx
004077F3    C64424 34 01       mov byte ptr ss:[esp+0x34],0x1
004077F8    E8 EF170900        call <jmp.&mfc42.#CString::CString_535>
004077FD    8D4424 18          lea eax,dword ptr ss:[esp+0x18]
00407801    885C24 30          mov byte ptr ss:[esp+0x30],bl;以上部分是取机器码等信息
```

```
00407805    50                 push eax
00407806    E8 65A3FFFF        call unpacked.00401B70 ;  注册验证部分
0040780B    83C4 0C            add esp,0xC
0040780E    8B0F               mov ecx,dword ptr ds:[edi]
00407810    C64424 28 02       mov byte ptr ss:[esp+0x28],0x2
00407815    8B41 F8            mov eax,dword ptr ds:[ecx-0x8]
00407818    83F8 04            cmp eax,0x4
0040781B    7F 35              jg short unpacked.00407852    ;用户名至少 5 位
0040781D    8D4C24 18          lea ecx,dword ptr ss:[esp+0x18]
00407821    E8 D8170900        call <jmp.&mfc42.#CString::CString_540>
00407826    6A 68              push 0x68
00407828    8D4C24 1C          lea ecx,dword ptr ss:[esp+0x1C]
0040782C    C64424 2C 03       mov byte ptr ss:[esp+0x2C],0x3
00407831    E8 84190900        call <jmp.&mfc42.#CString::LoadStringA_4160>
00407836    8B5424 18          mov edx,dword ptr ss:[esp+0x18]
0040783A    53                 push ebx
0040783B    53                 push ebx
0040783C    52                 push edx
0040783D    8BCE               mov ecx,esi
0040783F    E8 D81A0900        call <jmp.&mfc42.#CWnd::MessageBoxA_4224>
00407844    C64424 28 02       mov byte ptr ss:[esp+0x28],0x2
00407849    8D4C24 18          lea ecx,dword ptr ss:[esp+0x18]
0040784D    E9 CB000000        jmp unpacked.0040791D
00407852    8B4C24 10          mov ecx,dword ptr ss:[esp+0x10]
00407856    8B86 940E0000      mov eax,dword ptr ds:[esi+0xE94]
0040785C    51                 push ecx
0040785D    50                 push eax
0040785E    FF15 30495200      call dword ptr
ds:[<&msvcrt._mbscmp>]                          ; MSVCRT._mbscmp ;  真假注册码比较
00407864    83C4 08            add esp,0x8
00407867    85C0               test eax,eax
00407869    75 78              jnz short unpacked.004078E3    ;关键跳，跳就失败。
0040786B    891D F0164D00      mov dword ptr ds:[0x4D16F0],ebx
00407871    E8 A01A0900        call <jmp.&mfc42.#AfxGetModuleState_1168>
00407876    8B3F               mov edi,dword ptr ds:[edi]
00407878    8B68 04            mov ebp,dword ptr ds:[eax+0x4]
0040787B    57                 push edi
0040787C    68 D4534C00        push
unpacked.004C53D4                                       ; USER
00407881    68 C4534C00        push
unpacked.004C53C4                                       ; VisualToolbar10
00407886    8BCD               mov ecx,ebp
00407888    E8 831A0900        call <jmp.&mfc42.#CWinApp::WriteProfileStringA_6403>
0040788D    8B5424 10          mov edx,dword ptr ss:[esp+0x10]
```

```
00407891    8BCD              mov ecx,ebp
00407893    52                push edx
00407894    68 C0534C00       push
unpacked.004C53C0                                          ; SN
00407899    68 C4534C00       push
unpacked.004C53C4                                          ; VisualToolbar10
0040789E    E8 6D1A0900       call <jmp.&mfc42.#CWinApp::WriteProfileStringA_6403>
```

从上面的代码分析可以知道，程序取得输入的注册码，用户名和机器码之后，就会进入一个 call 里面进行运行，得出来的结果再直接和输入的注册码相比较，如果一样，就提示注册成功，否则失败。跟进这个关键 call，看它都干了些什么。

关键 call 代码：

```
00401B70    6A FF             push -0x1
00401B72    68 3F9F4900       push unpacked.00499F3F
00401B77    64:A1 00000000    mov eax,dword ptr fs:[0]
00401B7D    50                push eax
00401B7E    64:8925 0000000>mov dword ptr fs:[0],esp
00401B85    83EC 20           sub esp,0x20
00401B88    53                push ebx
00401B89    55                push ebp
00401B8A    56                push esi
00401B8B    33F6              xor esi,esi
00401B8D    57                push edi
00401B8E    897424 1C         mov dword ptr ss:[esp+0x1C],esi
00401B92    8D4C24 20         lea ecx,dword ptr ss:[esp+0x20]
00401B96    C74424 38 02000>mov dword ptr ss:[esp+0x38],0x2
00401B9E    E8 5B740900       call <jmp.&mfc42.#CString::CString_540>
00401BA3    8B4424 48         mov eax,dword ptr ss:[esp+0x48]
00401BA7    C64424 38 03      mov byte ptr ss:[esp+0x38],0x3
00401BAC    C64424 24 56      mov byte ptr ss:[esp+0x24],0x56
00401BB1    C64424 25 69      mov byte ptr ss:[esp+0x25],0x69
00401BB6    8B68 F8           mov ebp,dword ptr ds:[eax-0x8]
00401BB9    C64424 26 73      mov byte ptr ss:[esp+0x26],0x73
00401BBE    83FD 0A           cmp ebp,0xA              ; 用户名位数和 10 比较
00401BC1    C64424 27 75      mov byte ptr ss:[esp+0x27],0x75
00401BC6    C64424 28 61      mov byte ptr ss:[esp+0x28],0x61
00401BCB    C64424 29 6C      mov byte ptr ss:[esp+0x29],0x6C
00401BD0    C64424 2A 54      mov byte ptr ss:[esp+0x2A],0x54
00401BD5    C64424 2B 42      mov byte ptr ss:[esp+0x2B],0x42
00401BDA    C64424 2C 31      mov byte ptr ss:[esp+0x2C],0x31
00401BDF    C64424 2D 30      mov byte ptr ss:[esp+0x2D],0x30
00401BE4    7C 05             jl short unpacked.00401BEB    ; 如果大于 10 位，那么就分配
10 字节的空间。
00401BE6    BD 0A000000       mov ebp,0xA
```

```
00401BEB    55              push ebp            ; 否则根据用户名位数分配内存。
00401BEC    E8 1F740900     call <jmp.&mfc42.#operator new_823>
00401BF1    83C4 04         add esp,0x4
00401BF4    8D4C24 14       lea ecx,dword ptr ss:[esp+0x14]
00401BF8    8BF8            mov edi,eax
00401BFA    E8 FF730900     call <jmp.&mfc42.#CString::CString_540>
00401BFF    68 30164D00     push unpacked.004D1630
00401C04    8D4C24 14       lea ecx,dword ptr ss:[esp+0x14]
00401C08    C64424 3C 04    mov byte ptr ss:[esp+0x3C],0x4
00401C0D    E8 F8730900     call <jmp.&mfc42.#CString::CString_537>
00401C12    3BEE            cmp ebp,esi
00401C14    C64424 38 05    mov byte ptr ss:[esp+0x38],0x5
00401C19    7E 53           jle short unpacked.00401C6E
00401C1B    8D4424 24       lea eax,dword ptr ss:[esp+0x24]
00401C1F    2BC7            sub eax,edi
00401C21    894424 18       mov dword ptr ss:[esp+0x18],eax
00401C25    EB 04           jmp short unpacked.00401C2B
00401C27    8B4424 18       mov eax,dword ptr ss:[esp+0x18]
00401C2B    8B4C24 48       mov ecx,dword ptr ss:[esp+0x48]   ; esp+0x48 是用户名
00401C2F    8A140E          mov dl,byte ptr ds:[esi+ecx]      ; 逐位取用户名,存入 dl
00401C32    8B4C24 44       mov ecx,dword ptr ss:[esp+0x44] ; esp+0x44 是机器码
00401C36    8A1C0E          mov bl,byte ptr ds:[esi+ecx]; 逐位取机器码,存入 bl
00401C39    8D0C3E          lea ecx,dword ptr ds:[esi+edi]
00401C3C    8A0408          mov al,byte ptr ds:[eax+ecx] ; al 指向隐藏的字符串
"VisualTB10"
00401C3F    32C3            xor al,bl
00401C41    32C2            xor al,dl
00401C43    0FBED0          movsx edx,al   ; 用户名与机器码逐位异或，结果保存到 edx
中。
00401C46    8801            mov byte ptr ds:[ecx],al
00401C48    52              push edx
00401C49    8D4424 18       lea eax,dword ptr ss:[esp+0x18]
00401C4D    68 34504C00     push unpacked.004C5034            ; %d
00401C52    50              push eax
00401C53    E8 A0730900     call <jmp.&mfc42.#CString::Format_2818> ; 得到的值转化为
字符串形式。
00401C58    83C4 0C         add esp,0xC
00401C5B    8D4C24 14       lea ecx,dword ptr ss:[esp+0x14]
00401C5F    51              push ecx
00401C60    8D4C24 14       lea ecx,dword ptr ss:[esp+0x14]
00401C64    E8 9B730900     call <jmp.&mfc42.#CString::operator+=_939>   ; 把字符串连
接起来。
00401C69    46              inc esi
00401C6A    3BF5            cmp esi,ebp       ; 循环次数是用户名的位数
```

```
00401C6C   ^ 7C B9              jl short unpacked.00401C27
00401C6E     57                  push edi
00401C6F     E8 6C730900         call <jmp.&mfc42.#operator delete_825>
00401C74     8B7424 44           mov esi,dword ptr ss:[esp+0x44]
00401C78     83C4 04             add esp,0x4
00401C7B     8D5424 10           lea edx,dword ptr ss:[esp+0x10]
00401C7F     8BCE                mov ecx,esi
00401C81     52                  push edx
00401C82     E8 65730900         call <jmp.&mfc42.#CString::CString_535>
00401C87     BB 01000000         mov ebx,0x1
00401C8C     895C24 1C           mov dword ptr ss:[esp+0x1C],ebx
00401C90     8D4C24 10           lea ecx,dword ptr ss:[esp+0x10]
00401C94     C64424 38 04        mov byte ptr ss:[esp+0x38],0x4
00401C99     E8 48730900         call <jmp.&mfc42.#CString::~CString_800>
00401C9E     8D4C24 14           lea ecx,dword ptr ss:[esp+0x14]
00401CA2     C64424 38 03        mov byte ptr ss:[esp+0x38],0x3
00401CA7     E8 3A730900         call <jmp.&mfc42.#CString::~CString_800>
00401CAC     8D4C24 20           lea ecx,dword ptr ss:[esp+0x20]
00401CB0     C64424 38 02        mov byte ptr ss:[esp+0x38],0x2
00401CB5     E8 2C730900         call <jmp.&mfc42.#CString::~CString_800>
00401CBA     8D4C24 44           lea ecx,dword ptr ss:[esp+0x44]
00401CBE     885C24 38           mov byte ptr ss:[esp+0x38],bl
00401CC2     E8 1F730900         call <jmp.&mfc42.#CString::~CString_800>
00401CC7     8D4C24 48           lea ecx,dword ptr ss:[esp+0x48]
00401CCB     C64424 38 00        mov byte ptr ss:[esp+0x38],0x0
00401CD0     E8 11730900         call <jmp.&mfc42.#CString::~CString_800>
00401CD5     8B4C24 30           mov ecx,dword ptr ss:[esp+0x30]
00401CD9     8BC6                mov eax,esi
00401CDB     5F                  pop edi
00401CDC     5E                  pop esi
00401CDD     5D                  pop ebp
00401CDE     5B                  pop ebx
00401CDF     64:890D 0000000>mov dword ptr fs:[0],ecx
00401CE6     83C4 2C             add esp,0x2C
00401CE9     C3                  retn
```

注册过程很简单，就是机器码逐位异或用户名，用户名如果长度大于 10，那么只取前 10 位进行计算。对应的 WTL 实现代码如下：

```
DoDataExchange(DDX_SAVE); //从控件中取值

TCHAR szFlag[] = _T("VisualTB10");
TCHAR *p = szFlag;

m_strSN = _T(""); // m_strSN 保存生成的注册码
CString csTemp;
```

```
if (m_strMCode.IsEmpty() || m_strUserName.IsEmpty() ||
    (m_strUserName.GetLength() < 5))
    return 0;

int nLen = m_strUserName.GetLength();

TCHAR *pstrNameBuffer = new TCHAR[nLen<10?nLen+1:10];
TCHAR *pstrMCodeBuffer = new TCHAR[m_strMCode.GetLength()+1];

lstrcpy(pstrNameBuffer, m_strUserName.GetBuffer(m_strUserName.GetLength()));
lstrcpy(pstrMCodeBuffer, m_strMCode.GetBuffer(m_strMCode.GetLength()));

for (int i = 0; i < (nLen<10?nLen:10); i++)
{
    csTemp.Format(_T("%d"), p[i] ^ pstrMCodeBuffer[i] ^ pstrNameBuffer[i]);
    m_strSN += csTemp;
}

DoDataExchange(DDX_LOAD); //显示到控件中。
```

注册机就是这么简单，编译运行，把生成的注册码填到程序中，程序已经提示注册成功了，要重启程序才能生效，可以重启程序之后发现，程序竟然还是未注册版本！哎呀呀，被程序戏弄了。

既然程序要重启验证，那么验证注册码的地方肯定在程序初始化的部分进行，如果验证成功，那么程序就会把标题栏的未注册信息去掉，好了，带着这个猜想，下断 bp SetWindowTextA

运行后程序断下来了，看来猜想是正确的，往上找到关键部分:

```
0041BC54    BB 01000000     mov ebx,0x1
0041BC59    68 C4534C00     push
unpacked.004C53C4                                   ; VisualToolbar10
0041BC5E    8BCE            mov ecx,esi
0041BC60    891D F0164D00   mov dword ptr ds:[0x4D16F0],ebx
0041BC66    E8 57DC0700     call <jmp.&mfc42.#CWinApp::SetRegistryKey_6117>
0041BC6B    8D4424 08       lea eax,dword ptr ss:[esp+0x8]
0041BC6F    50              push eax
0041BC70    E8 EB5DFEFF     call unpacked.00401A60              ; 生成机器码
0041BC75    83C4 04         add esp,0x4
0041BC78    68 D4534C00     push
unpacked.004C53D4                                   ; USER
0041BC7D    68 D4534C00     push
unpacked.004C53D4                                   ; USER
0041BC82    8D4C24 14       lea ecx,dword ptr ss:[esp+0x14]
0041BC86    68 C4534C00     push
unpacked.004C53C4                                   ; VisualToolbar10
```

```
0041BC8B    51                  push ecx
0041BC8C    8BCE                mov ecx,esi
0041BC8E    C78424 CC480000>mov dword ptr ss:[esp+0x48CC],0x0
0041BC99    E8 1EDC0700         call <jmp.&mfc42.#CWinApp::GetProfileStringA_3522>
0041BC9E    68 DC5A4C00         push
unpacked.004C5ADC                                       ; Password
0041BCA3    68 C0534C00         push
unpacked.004C53C0                                       ; SN
0041BCA8    8D5424 20           lea edx,dword ptr ss:[esp+0x20]
0041BCAC    68 C4534C00         push
unpacked.004C53C4                                       ; VisualToolbar10
0041BCB1    52                  push edx
0041BCB2    8BCE                mov ecx,esi
0041BCB4    889C24 CC480000 mov byte ptr ss:[esp+0x48CC],bl
0041BCBB    E8 FCDB0700         call <jmp.&mfc42.#CWinApp::GetProfileStringA_3522>
0041BCC0    51                  push ecx
0041BCC1    8D4424 10           lea eax,dword ptr ss:[esp+0x10]
0041BCC5    8BCC                mov ecx,esp
0041BCC7    896424 20           mov dword ptr ss:[esp+0x20],esp
0041BCCB    50                  push eax
0041BCCC    C68424 C4480000>mov byte ptr ss:[esp+0x48C4],0x2
0041BCD4    E8 13D30700         call <jmp.&mfc42.#CString::CString_535>
0041BCD9    51                  push ecx
0041BCDA    8D5424 10           lea edx,dword ptr ss:[esp+0x10]
0041BCDE    8BCC                mov ecx,esp
0041BCE0    896424 1C           mov dword ptr ss:[esp+0x1C],esp
0041BCE4    52                  push edx
0041BCE5    C68424 C8480000>mov byte ptr ss:[esp+0x48C8],0x3
0041BCED    E8 FAD20700         call <jmp.&mfc42.#CString::CString_535>
0041BCF2    8D4424 18           lea eax,dword ptr ss:[esp+0x18]
0041BCF6    C68424 C4480000>mov byte ptr ss:[esp+0x48C4],0x2
0041BCFE    50                  push eax
0041BCFF    E8 6C5EFEFF         call unpacked.00401B70 ;注册验证的部份，在前面已经分析过
了。
0041BD04    8B4C24 24           mov ecx,dword ptr ss:[esp+0x24]
0041BD08    8B5424 1C           mov edx,dword ptr ss:[esp+0x1C]
0041BD0C    51                  push ecx
0041BD0D    52                  push edx
0041BD0E    C68424 D0480000>mov byte ptr ss:[esp+0x48D0],0x4
0041BD16    FF15 30495200       call dword ptr ds:[<&msvcrt._mbscmp>]        ;
MSVCRT._mbscmp  ; 真假码比较部分。
0041BD1C    83C4 14             add esp,0x14
0041BD1F    85C0                test eax,eax
0041BD21    75 07               jnz short unpacked.0041BD2A; 关键跳
```

在注册成功的时候，程序已经把注册信息保存到注册表中了，程序会在初始化的时候对注册信息进行验证。最可疑的地方是 0041BC70 处的 call 跟进行看它做了什么：

```
00401A60    6A FF               push -0x1
00401A62    68 EF9E4900         push unpacked.00499EEF
00401A67    64:A1 00000000      mov eax,dword ptr fs:[0]
00401A6D    50                  push eax
00401A6E    64:8925 0000000>mov dword ptr fs:[0],esp
00401A75    83EC 10             sub esp,0x10
00401A78    56                  push esi
00401A79    57                  push edi
00401A7A    C74424 14 00000>mov dword ptr ss:[esp+0x14],0x0
00401A82    E8 A9FFFFFF         call unpacked.00401A30          ; 取 C 盘序列号并简单处理
00401A87    8BF8                mov edi,eax
00401A89    B8 8F588B4F         mov eax,0x4F8B588F          ; 64 位乘法,结果保存形式为
EDX:EAX
00401A8E    F7E7                mul edi                     ; edi 的值不变
00401A90    8BF7                mov esi,edi
00401A92    8D4C24 0C           lea ecx,dword ptr ss:[esp+0xC]
00401A96    2BF2                sub esi,edx                     ; edx 是高 32 位
00401A98    D1EE                shr esi,1
00401A9A    03F2                add esi,edx
00401A9C    C1EE 10             shr esi,0x10
00401A9F    E8 5A750900         call <jmp.&mfc42.#CString::CString_540>
00401AA4    56                  push esi
00401AA5    8D4424 10           lea eax,dword ptr ss:[esp+0x10]
00401AA9    68 34504C00         push unpacked.004C5034              ; %d
00401AAE    50                  push eax
00401AAF    C74424 2C 01000>mov dword ptr ss:[esp+0x2C],0x1
00401AB7    E8 3C750900         call <jmp.&mfc42.#CString::Format_2818> ; 格式化为字符
串。
00401ABC    8D04B6              lea eax,dword ptr ds:[esi+esi*4]     ; 等价于 eax=esi*5
00401ABF    83C4 0C             add esp,0xC
00401AC2    8D0480              lea eax,dword ptr ds:[eax+eax*4]        ; eax=eax*5
00401AC5    8D0480              lea eax,dword ptr ds:[eax+eax*4]        ; eax=eax*5
00401AC8    8D0480              lea eax,dword ptr ds:[eax+eax*4]        ; eax=eax*5
00401ACB    8D0C80              lea ecx,dword ptr ds:[eax+eax*4]        ; ecx=eax*5
00401ACE    C1E1 05             shl ecx,0x5
00401AD1    2BF9                sub edi,ecx                         ; edi 的值还是上面的。
00401AD3    8D4C24 08           lea ecx,dword ptr ss:[esp+0x8]
00401AD7    E8 22750900         call <jmp.&mfc42.#CString::CString_540>
00401ADC    57                  push edi
00401ADD    8D5424 0C           lea edx,dword ptr ss:[esp+0xC]
00401AE1    68 34504C00         push unpacked.004C5034              ; %d
```

```
00401AE6    52                  push edx
00401AE7    C64424 2C 02        mov byte ptr ss:[esp+0x2C],0x2
00401AEC    E8 07750900         call <jmp.&mfc42.#CString::Format_2818>   ; 格式化字符串
00401AF1    83C4 0C             add esp,0xC
00401AF4    8D4424 08           lea eax,dword ptr ss:[esp+0x8]
00401AF8    8D4C24 0C           lea ecx,dword ptr ss:[esp+0xC]
00401AFC    8D5424 10           lea edx,dword ptr ss:[esp+0x10]
00401B00    50                  push eax
00401B01    51                  push ecx
00401B02    52                  push edx
00401B03    E8 EA740900         call <jmp.&mfc42.#operator+_922>   ; 把上面的两个字符串
连接起来，得到机器码
00401B08    8B7424 28           mov esi,dword ptr ss:[esp+0x28]
00401B0C    8D4424 10           lea eax,dword ptr ss:[esp+0x10]
00401B10    50                  push eax
00401B11    8BCE                mov ecx,esi
00401B13    C64424 24 03        mov byte ptr ss:[esp+0x24],0x3
00401B18    E8 CF740900         call <jmp.&mfc42.#CString::CString_535>
00401B1D    C74424 14 01000>mov dword ptr ss:[esp+0x14],0x1
00401B25    8D4C24 10           lea ecx,dword ptr ss:[esp+0x10]
00401B29    C64424 20 02        mov byte ptr ss:[esp+0x20],0x2
00401B2E    E8 B3740900         call <jmp.&mfc42.#CString::~CString_800>
00401B33    8D4C24 08           lea ecx,dword ptr ss:[esp+0x8]
00401B37    C64424 20 01        mov byte ptr ss:[esp+0x20],0x1
00401B3C    E8 A5740900         call <jmp.&mfc42.#CString::~CString_800>
00401B41    8D4C24 0C           lea ecx,dword ptr ss:[esp+0xC]
00401B45    C64424 20 00        mov byte ptr ss:[esp+0x20],0x0
00401B4A    E8 97740900         call <jmp.&mfc42.#CString::~CString_800>
00401B4F    8B4C24 18           mov ecx,dword ptr ss:[esp+0x18]
00401B53    8BC6                mov eax,esi
00401B55    5F                  pop edi
00401B56    5E                  pop esi
00401B57    64:890D 0000000>mov dword ptr fs:[0],ecx
00401B5E    83C4 1C             add esp,0x1C
00401B61    C3                  retn
```

从上面的分析可以知道，这个 call 的作用就是生成一个机器码。相应的 WTL 代码如下：

```
CString csM1, csM2;

DWORD dwVSN, dwEDI;
GetVolumeInformation(_T("C:\\"), NULL, 0xC, &dwVSN, NULL, NULL, NULL, 0xA);

dwEDI = dwVSN ^= 0x18915791; //edi
```

```
DWORD64 dw64Number = (DWORD64)dwVSN * 0x4f8b588f;
DWORD dwTemp = HIDWORD(dw64Number);

dwVSN -= dwTemp;
dwVSN >>= 1;
dwVSN += dwTemp;
dwVSN >>= 0x10;

csM1.Format(_T("%d"), dwVSN);

DWORD dwTemp2 = dwVSN * 5;
dwTemp2 *= 5;
dwTemp2 *= 5;
dwTemp2 *= 5;

DWORD dwTemp3 = dwTemp2 *5;
dwTemp3 <<= 5;
dwEDI -= dwTemp3;

csM2.Format(_T("%d"), dwEDI);

m_strMCode = csM1 + csM2; //生成的机器码

DoDataExchange(DDX_LOAD);
```

既然注册验证部分都是一样的，为什么验证不了呢，程序明明已经提示注册成功了呀？看了猫腻就在这个机器码身上，编译运行看一下验证用的机器码和在注册界面里显示的机器码有什么不一样：

验证用的机器码: 67206293

注册界面里显示的机器码: 672006293

看到差别了么？可恶呀，在中间插了一个 0。

现在明白了，程序的注册验证流程是这样的，用户输入用户名，和序列号。如果注册成功，就提示要重启程序进行验证，然后程序在重启验证过程中，用来参与验证的机器码和在注册界面时显示的机器码不一样，所以即使猜解了注册算法，做出了注册机，只能是让程序提示注册成功信息而已，还要找出程序生成机器码的部份，用真正的机器码来生成注册码才能真正让程序注册。

作者应该是根据用户邮寄回来的机器码和用户名来计算真正的注册码返还给用户，用户在注册的时候，除了输入用户名和注册码，还要输入机器码，这也是为什么机器码那一栏是可以输入的原因了。

举个例子，假如我要向作者购买该软件授权，那么我需要把机器码，用户名发给作者：

机器码：672006293 (在注册界面显示的)

用户名：cntrump (这是我要注册的用户名)

作者会回复下列信息：

机器码：67206293 (用上面的机器码计算得到，根据对比发现只是简单的去掉中间的 0)
用户名：cntrump
序列号：3485355345129

我需要把这三项都填上才能成功注册。

要做出和作者一样的注册机，还需要把 用户机器码 -> 注册机器码 这部分分析出来，根据上面的分析已经知道了注册机器码的生成过程，通过和在注册界面上显示的机器码对比可以发现，用户机器码去掉中间的 0 就等于注册机器码了。相应的 WTL 代码如下：

```
CString GetRealMachineID(CString csUserMachineID)
{
    int nLen = csUserMachineID.GetLength();
    CString csM1 = csUserMachineID.Left(nLen/2);    // 取前半部分
    CString csM2 = csUserMachineID.Right(nLen/2);    // 取后半部分

    return csM1+csM2; //前后两部分结合就是注册机器码
}
```

有了这些信息，就能写出和作者一模一样的注册机了，核心代码如下（WTL）：

```
    DoDataExchange(DDX_SAVE);
//和上面生成注册码不同的地方就是多了下面这 2 句代码。
    CString csTmpMCode = m_strMCode;    //用中间变量保存用户机器码
    m_strMCode = GetRealMachineID(csTmpMCode); //计算得到注册机器码

    TCHAR szFlag[] = _T("VisualTB10");
    TCHAR *p = szFlag;

    m_strSN = _T("");
    CString csTemp;

    if (m_strMCode.IsEmpty() ||
        m_strUserName.IsEmpty() ||
        (m_strUserName.GetLength() < 5))
        return 0;

    int nLen = m_strUserName.GetLength();

    TCHAR *pstrNameBuffer = new TCHAR[nLen<10?nLen+1:10];
    TCHAR *pstrMCodeBuffer = new TCHAR[m_strMCode.GetLength()+1];

    lstrcpy(pstrNameBuffer, m_strUserName.GetBuffer(m_strUserName.GetLength()));
    lstrcpy(pstrMCodeBuffer, m_strMCode.GetBuffer(m_strMCode.GetLength()));

    for (int i = 0; i < (nLen<10?nLen:10); i++)
    {
        csTemp.Format(_T("%d"), p[i] ^ pstrMCodeBuffer[i] ^ pstrNameBuffer[i]);
        m_strSN += csTemp;
```

```
}

    DoDataExchange(DDX_LOAD); //把最终结果显示到界面。
```

顺便说一下，代码虽然使用 WTL，但是也能换成 MFC 的代码，只需要把 DoDataExchange 换成 UpdateData，DDX_LOAD 换成 FALSE,DDX_SAVE 换成 TRUE，就可以了。