关于 XP SP3 系统下 NtUserCreateWindowEx 原型分析
修正版
By HSQ

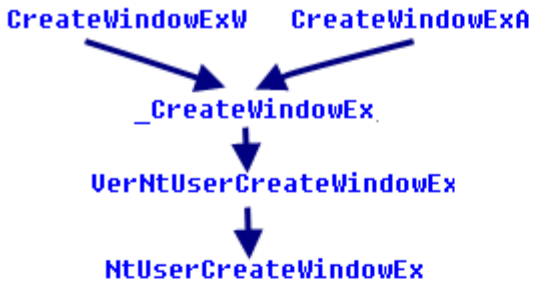
说明：其实以前就因为需要用到这个函数来处理一些特定窗口，苦于没有该 API 原型，最后自己也是草草的 DBG 了一下，可弄错来的结果还是有偏差。最近再次用那些东西试试效果，居然发现用起来完全形同虚设。可能是系统升级后，原来蒙的偏移量改变了。乘着闲来无事，干脆下决心给彻底弄清楚其原型。顺便将分析过程记录存档，以便共享后来者和遗忘是查阅。

1. 其最终由 WIN32K.SYS 实现

IDA - D:\IDA systwm32\win32k\5.1.2600.5756\win32k.sys - [IDA View-A]

```
.text:BF82F953 ; int __stdcall NtUserCreateWindowEx(char, __int16, ULONG NumberOfBytes, int, int, int
.text:BF82F953 __stdcall NtUserCreateWindowEx(x, x, x, x, x, x, x, x, x, x, x, x, x, x, x) proc near
.text:BF82F953                                         ; DATA XREF: .data:BF99A65C↓o
```

2. 调用流程



user32.dll

```
__stdcall NtUserCreateWindowEx(x, x, x, >
                                       ;
                mov     eax, 4439
                mov     edx, 7FFE0300h
                call    dword ptr [edx]

                retn    3Ch

__stdcall NtUserCreateWindowEx(x, x, x, >
```

模块 - user32]

```
77D2E37D   B8 57110000   mov     eax, 1157
77D2E382   BA 0003FE7F   mov     edx, 7FFE0300
77D2E387   FF12          call    dword ptr [edx]   ntdll.KiFastSystemCall
77D2E389   C2 3C00       retn    3C
```

模块 - ntdll]

```
7C92E4E6   8DA424 0000000( lea     esp, dword ptr [esp]
7C92E4ED   8D49 00         lea     ecx, dword ptr [ecx]
7C92E4F0   8BD4            mov     edx, esp
7C92E4F2   0F34            sysenter
7C92E4F4   C3              retn
```

3. 参数传递经过

3.1 CreateWindowEx（CreateWindowExW/A）

先从 2k 源码看参数传递过程开始分析

```
HWND WINAPI CreateWindowEx(
    DWORD dwExStyle,
    LPCTSTR lpClassName,
    LPCTSTR lpWindowName,
    DWORD dwStyle,
    int X,
    int Y,
    int nWidth,
    int nHeight,
    HWND hWndParent,
    HMENU hMenu,
    HINSTANCE hModule,
    LPVOID lpParam)
```

```
#ifdef UNICODE
#define IS_ANSI FALSE
#ifndef _UNICODE
#define _UNICODE
#endif
#else
#define IS_ANSI TRUE
#undef _UNICODE
#endif
```

```
return _CreateWindowEx(dwExStyle, lpClassName, lpWindowName,
        dwStyle, X, Y, nWidth, nHeight, hWndParent, hMenu,
        hModule, lpParam, IS_ANSI);
```

注意：在 2K 中没有 CreateWindowExW/A 之分,这也正是需要对 NtUserCreateWindowEx 进行重新分析的原因
CreateWindowEx 没有对参数进行任何处理，而是追加一个参数后直接调用_CreateWindowEx

3.2 _CreateWindowEx

```
HWND _CreateWindowEx(
    DWORD dwExStyle,
    LPCTSTR pClassName,
    LPCTSTR pWindowName,
    DWORD dwStyle,
    int x,
    int y,
    int nWidth,
    int nHeight,
    HWND hwndParent,
    HMENU hmenu,
    HANDLE hModule,
    LPVOID pParam,
    DWORD dwFlags)
```

可见，在 2k 时，是通过是否定义了 #ifdef UNICODE 这个标志来调用相应版本的 API，及 dwFlags 来确定是 W/A 之分的。而在 XP 中，已经存 CreateWindowExW/A 之分了，在通过 IDA 分析可知

对于 CreateWindowExW 的调用：

```
push    40000000h        ; dwFlags
push    [ebp+lpParam]    ; lpParam
push    [ebp+hInstance]  ; hInstance
push    [ebp+hMenu]      ; hMenu
push    [ebp+hWndParent] ; hWndParent
push    [ebp+nHeight]    ; nHeight
push    [ebp+nWidth]     ; nWidth
push    [ebp+Y]          ; Y
push    [ebp+X]          ; X
push    [ebp+dwStyle]    ; dwStyle
push    [ebp+lpWindowName] ; lpWindowName
push    [ebp+lpClassName] ; lpClassName
push    dword ptr [ebp+dwExStyle] ; dwExStyle
call     CreateWindowEx(x,x,x,x,x,x,x,x,x,x,x,x
```

对于 CreateWindowExA 的调用：

```
push    40000001h        ; dwFlags
push    [ebp+lpParam]    ; lpParam
push    [ebp+hInstance]  ; hInstance
push    [ebp+hMenu]      ; hMenu
push    [ebp+hWndParent] ; hWndParent
push    [ebp+nHeight]    ; nHeight
push    [ebp+nWidth]     ; nWidth
push    [ebp+Y]          ; Y
push    [ebp+X]          ; X
push    [ebp+dwStyle]    ; dwStyle
push    [ebp+lpWindowName] ; lpWindowName
push    [ebp+lpClassName] ; lpClassName
push    dword ptr [ebp+dwExStyle] ; dwExStyle
call    _CreateWindowEx(x,x,x,x,x,x,x,x,x,x,x,x
```

即 A 版本的dwFlags= 40000001h，W 版本为40000000h，其他参数都沿袭 2K。

_CreateWindowEx 进行一些处理后接着开始进入下一层调用，这里 XP 与 2K 处理有些差异。

3.3 NtUserCreateWindowEx（VerNtUserCreateWindowEx）

A. 在 2K 中，到次就直接进入了 NtUserCreateWindowEx 流程

```
retval = (ULONG_PTR)NtUserCreateWindowEx(
      dwExStyle,
      pstrClassName,
      pstrWindowName,
      dwStyle,
      x,
      y,
      nWidth,
      nHeight,
      hwndParent,
      hmenu,
      hModule,
      pParam,
      dwExpWinVerAndFlags);
```

NtUserCreateWindowEx 的原型：

```
HWND NtUserCreateWindowEx(
   IN DWORD dwExStyle,
   IN PLARGE_STRING pstrClassName,
   IN PLARGE_STRING pstrWindowName OPTIONAL,
   IN DWORD dwStyle,
   IN int x,
   IN int y,
   IN int nWidth,
   IN int nHeight,
   IN HWND hwndParent,
   IN HMENU hmenu,
   IN HANDLE hModule,
   IN LPVOID pParam,
   IN DWORD dwFlags)
```

至此，用户层的处理已经结束。

B.而在 XP 中，却还得经过一个 VerNtUserCreateWindowEx 调用才会进入 NtUserCreateWindowEx 流程。结合 2K 的源码，拿 IDA 的 F5 初步分析，猜测 VerNtUserCreateWindowEx 应该就是以前 NtUserCreateWindowEx 的伪函数

```
v21 = lpWindowName;
v22 = edi0;
v20 = lpClassName;

RtlCaptureLargeAnsiString(&v32, v20, 1);
v20 = v33;
RtlInitLargeUnicodeString((int)&v32, v20, 0xFFFFFFFFu);
v20 = (const WCHAR *)&v32;
        RtlInitLargeAnsiString(&plstr, v21, -1);
LABEL_20:
        lpWindowName = (LPCWSTR)&plstr;
LABEL_10:
        v24 = (HWND)VerNtUserCreateWindowEx(
                        dwExStyle,
                        v20,
                        lpWindowName,
                        dwStyle,
                        X,
                        Y,
                        nWidth,
                        nHeight,
                        hWndParent,
                        hMenu,
                        hInstance,
                        lpParam,
                        (unsigned int)v30 | v23 & 0xC0000000);
```

再看看该处使用到的一些字符结构

```
typedef struct _LARGE_STRING {
    ULONG Length;
    ULONG MaximumLength : 31;
    ULONG bAnsi : 1;
    PVOID Buffer;
} LARGE_STRING, *PLARGE_STRING;

typedef struct _LARGE_ANSI_STRING {
    ULONG Length;
    ULONG MaximumLength : 31;
    ULONG bAnsi : 1;
    PSTR Buffer;
} LARGE_ANSI_STRING, *PLARGE_ANSI_STRING;

typedef struct _LARGE_UNICODE_STRING {
    ULONG Length;
    ULONG MaximumLength : 31;
    ULONG bAnsi : 1;
    PWSTR Buffer;
} LARGE_UNICODE_STRING, *PLARGE_UNICODE_STRING;
```

由此可见 PLARGE_STRING 是 PLARGE_ANSI_STRING 与 PLARGE_UNICODE_STRING 的通用结构，而 VerNtUserCreateWindowEx 内代码片无非是根据上层函数传入的dwFlags 进行做 A/W 的配对吻合而已，故最终确定其原型应该为

```
HWND VerNtUserCreateWindowEx(
  IN DWORD dwExStyle,
  IN PLARGE_STRING pstrClassName,
  IN PLARGE_STRING pstrWindowName OPTIONAL,
  IN DWORD dwStyle,
  IN int x,
  IN int y,
  IN int nWidth,
  IN int nHeight,
  IN HWND hwndParent,
  IN HMENU hmenu,
  IN HANDLE hModule,
  IN LPVOID pParam,
  IN DWORD dwFlags)
```

在此基础上，用 IDA 进行分析看得出 NtUserCreateWindowEx 的最终原型基本为，

```
xor     eax, eax
push    ebx             ; OUT PSIZE_T ReturnLength OP
mov     [ebp+lstrActivationContextInformation], ebx
lea     edi, [ebp+lpstrClassName]
stosd
push    8               ; IN SIZE_T ActivationContext
lea     eax, [ebp+lstrActivationContextInformation]
push    eax             ; OUT PVOID ActivationContext
push    1               ; IN ACTIVATION_CONTEXT_INFO_
call    ds:RtlQueryInformationActiveActivationContext

push    [ebp+pActivationContextInformation]
push    [ebp+dwFlags]
push    [ebp+lpParam]
push    [ebp+lhModule]
push    [ebp+lhmenu]
push    [ebp+lhwndParent]
push    [ebp+nHeight]
push    [ebp+nWidth]
push    [ebp+y]
push    [ebp+x]
push    [ebp+dwStyle]
push    [ebp+lpstrWindowName]
push    [ebp+lpstrClassName]
push    esi
push    [ebp+dwExStyle]
call    NtUserCreateWindowEx(x,x,x,x,x,x,x,x,x,x,x,x,x,x,x)
```

4. 最终整理 XP SP3 下得到真正的 NtUserCreateWindowEx

```
HWND NtUserCreateWindowEx(
        IN DWORD dwExStyle,
        IN DWORD dwMaybeClassVer,
        IN PLARGE_UNICODE_STRING pstrClassName,
        IN PLARGE_UNICODE_STRING pstrWindowName OPTIONAL,
        IN DWORD dwStyle,
        IN int x,
        IN int y,
        IN int nWidth,
        IN int nHeight,
        IN HWND hWndParent,
        IN HMENU hMenu,
        IN HANDLE hModule,
        IN LPVOID pParam,
        IN DWORD dwFlags,
        IN PVOID pActivationContextInformation)
```

由于昨晚马虎，依据以前的经验，直接认为传到内核后所有的字符参数应该均为宽字节类型，于是直接将 pstrClassName 和 pstrWindowName 参数类型设为 PLARGE_UNICODE_STRING。后来在自己写个创建窗口的例子时，发现用 A 版 CreateWindowEx 创建了一个自定义的类窗口，到内核层却只能把窗口类名称给解析出

来。再看看以前的分析过程，于是猜测也可能存在传到内核还是单字节的情况，随即在自己的代码内加了 **W/A** 之分后，发现窗口的类好名都能很好的解析出来。于是它们的参数类型应该为的 **PLARGE_STRING**，自己在内核里还得具体区别对待。修正后的原型如下：

```
NTSTATUS NtUserCreateWindowEx(
        IN DWORD dwExStyle,
        IN DWORD dwMaybeClassVer,
        IN PLARGE_STRING pstrClassName,
        IN PLARGE_STRING pstrWindowName OPTIONAL
        IN DWORD dwStyle,
        IN int x,
        IN int y,
        IN int nWidth,
        IN int nHeight,
        IN HWND hWndParent,
        IN HMENU hMenu,
        IN HANDLE hModule,
        IN LPVOID pParam,
        IN DWORD dwFlags,
        IN PVOID pActivationContextInformation)
```

5. 验证分析结果正确性

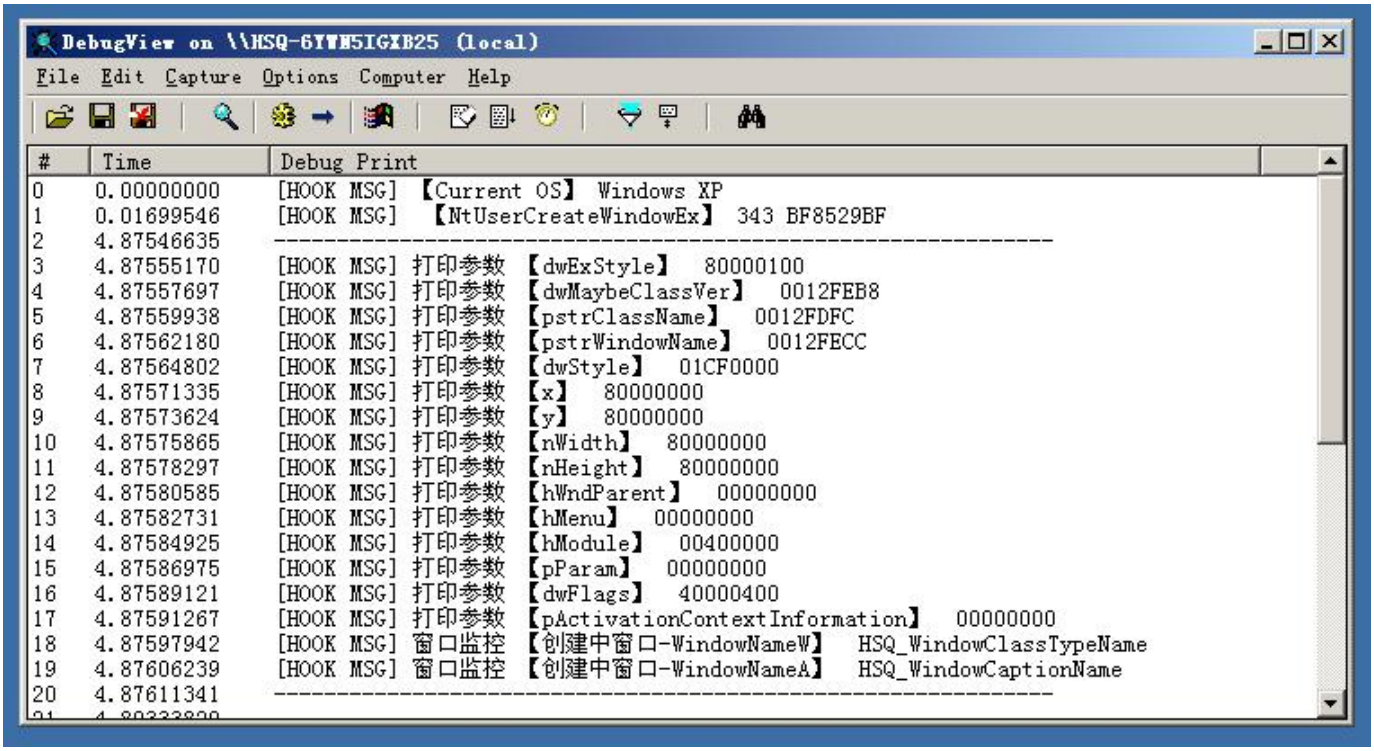
测试例子的代码片：

```
                .const
szWinMainCaption        db "HSQ_WindowCaptionName",0
szWinMainClassName      db "HSQ_WindowClassTypeName",0

 mov     wc.cbSize,sizeof WNDCLASSEX
 mov     wc.style,CS_HREDRAW or CS_VREDRAW or CS_DBLCLKS
 mov     wc.lpfnWndProc,WinMainProc
 mov     wc.cbClsExtra,NULL
 mov     wc.cbWndExtra,NULL
 push    hInst
 pop     wc.hInstance
 mov     wc.hCursor,eax
 mov     wc.hIcon,eax
 mov     wc.hbrBackground,COLOR_BACKGROUND
 mov     wc.lpszMenuName,NULL
 mov     wc.lpszClassName,offset szWinMainClassName

 lea     eax, wc
 push    eax
 call    RegisterClassEx

 push    0
 push    hInst
 push    0
 push    0
 push    CW_USEDEFAULT
 push    CW_USEDEFAULT
 push    CW_USEDEFAULT
 push    CW_USEDEFAULT
 push    WS_MAXIMIZE or WS_OVERLAPPEDWINDOW
 push    offset szWinMainCaption
 push    offset szWinMainClassName
 push    WS_EX_WINDOWEDGE
 call    CreateWindowEx
```

测试驱动的解析代码片：

```
if(pstrClassName)
{
    if (pstrClassName->bAnsi)
    {
        RtlInitAnsiString(&aString,(PCSZ)pstrClassName->Buffer);
        RtlAnsiStringToUnicodeString(&usTemp,&aString,TRUE);
        DbgPrint("[HOOK MSG] 窗口监控 【创建中窗口-WindowNameA】   %wZ\n",&usTemp);
        RtlFreeUnicodeString(&usTemp);
    }
    else
    {
        RtlInitUnicodeString(&usTemp, (PWSTR)pstrClassName->Buffer);
        DbgPrint("[HOOK MSG] 窗口监控 【创建中窗口-WindowNameW】   %wZ\n",&usTemp);
    }
}
if(pstrWindowName)
{
    if (pstrWindowName->bAnsi)
    {
        RtlInitAnsiString(&aString,(PCSZ)pstrWindowName->Buffer);
        RtlAnsiStringToUnicodeString(&usTemp,&aString,TRUE);
        DbgPrint("[HOOK MSG] 窗口监控 【创建中窗口-WindowNameA】   %wZ\n",&usTemp);
        RtlFreeUnicodeString(&usTemp);
    }
    else
    {
        RtlInitUnicodeString(&usTemp, (PWSTR)pstrWindowName->Buffer);
        DbgPrint("[HOOK MSG] 窗口监控 【创建中窗口-WindowNameW】   %wZ\n",&usTemp);
    }
}
```

最后测试了一下，证明这次分析的是完全对的...^_^....



参考资料：

1. **<<WRK 1.2>> 部分源码**
2. **<< windows_2000_source_code>> 部分源码**

2009 年 3 月 13 日星期五(北京)