获取Kernel32基地址的几种方法-相关结构

转载

爱沧海 于 2013-04-01 17:20:55 发布 4101 收藏

一、几个重要的数据结构，可以通过windbg的dt命令查看其详细信息

_PEB、_PEB_LDR_DATA、_LDR_DATA_TABLE_ENTRY

二、技术原理

1、通过fs:[30h]获取当前进程的_PEB结构

2、通过_PEB的Ldr成员获取_PEB_LDR_DATA结构

3、通过_PEB_LDR_DATA的InMemoryOrderModuleList成员获取_LIST_ENTRY结构

4、通过_LIST_ENTRY的Flink成员获取_LDR_DATA_TABLE_ENTRY结构，注意：这里的Flink指向的是_LDR_DATA_TABLE_ENTRY结构中的InMemoryOrderLinks成员，因此需要计算真正的_LDR_DATA_TABLE_ENTRY起始地址，我们可以用CONTAINING_RECORD来计算。

5、输出_LDR_DATA_TABLE_ENTRY的BaseDllName或FullDllName成员信息。

三、代码实现(基于XP sp2 系统)

```
//EnumInLoadModule.c
//compile:cl EnumInLoadModule.c
#include <windows.h>

#define CONTAINING_RECORD(address, type, field) ((type *)( /
(PCHAR)(address) - /
(ULONG_PTR)(&((type *)0)->field)))

typedef struct _UNICODE_STRING {
    USHORT  Length;
    USHORT  MaximumLength;
    PWSTR  Buffer;
} UNICODE_STRING, *PUNICODE_STRING;

typedef struct _PEB_LDR_DATA
{
    DWORD Length;
    UCHAR Initialized;
    PVOID SsHandle;
    LIST_ENTRY InLoadOrderModuleList;
    LIST_ENTRY InMemoryOrderModuleList;
    LIST_ENTRY InInitializationOrderModuleList;
    PVOID EntryInProgress;
}PEB_LDR_DATA,*PPEB_LDR_DATA;

typedef struct _LDR_DATA_TABLE_ENTRY
{
    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;
    DWORD SizeOfImage;
    UNICODE_STRING FullDllName;
    UNICODE_STRING BaseDllName;
    DWORD Flags;
    WORD LoadCount;
    WORD TlsIndex;
    LIST_ENTRY HashLinks;
    PVOID SectionPointer;
    DWORD CheckSum;
    DWORD TimeDateStamp;
    PVOID LoadedImports;
    PVOID EntryPointActivationContext;
    PVOID PatchInformation;
}LDR_DATA_TABLE_ENTRY,*PLDR_DATA_TABLE_ENTRY;
```

其中，InLoadOrderModuleList(xp下第一个结点XXX.exe 第二个是ntdll.dll，win7:ntdll.dll,kernel32.dll)、InMemoryOrderModuleList（1.xx.exe 2.ntdll.dll win7:xx.exe,ntdll.dll）、InInitializationOrderModuleList(xp下第一个结点一般是ntdll.dll，第二个是 kernel32.dll win7:ntdll,kernelbase.dll)就是进程当前已加载模块的链表，只是按照不同的方式排序。EnumProcessModules是通过InMemoryOrderModuleList链表枚举的，而根据Win2k代码，ToolHelp32函数是通过InLoadOrderModuleList枚举。这三个_LIST_ENTRY都是在一个RTL_PROCESS_MODULE_INFORMATION结构中的成员。这个结构在2k代码中有引用，不过没有确切的定义

```
typedef struct _PEB
{
    UCHAR InheritedAddressSpace;
    UCHAR ReadImageFileExecOptions;
    UCHAR BeingDebugged;
    UCHAR SpareBool;
    PVOID Mutant;
    PVOID ImageBaseAddress;
    PPEB_LDR_DATA Ldr;


    //......
}PEB,*PPEB;

int main(void)
{
    PLDR_DATA_TABLE_ENTRY pLdrDataEntry = NULL;
    PLIST_ENTRY pListEntryStart = NULL,pListEntryEnd = NULL;
    PPEB_LDR_DATA pPebLdrData = NULL;
    PPEB pPeb = NULL;

    //故意加载一些DLL，以便测试!
    LoadLibrary("ResLibDemo");
    __asm
    {
        //1、通过fs:[30h]获取当前进程的_PEB结构
        mov eax,dword ptr fs:[30h];
        mov pPeb,eax
    }

    //2、通过_PEB的Ldr成员获取_PEB_LDR_DATA结构
    pPebLdrData = pPeb->Ldr;

    //3、通过_PEB_LDR_DATA的InMemoryOrderModuleList成员获取_LIST_ENTRY结构
    pListEntryStart = pListEntryEnd = pPebLdrData->InMemoryOrderModuleList.Flink;

    //查找所有已载入到内存中的模块
    do
    {
        //4、通过_LIST_ENTRY的Flink成员获取_LDR_DATA_TABLE_ENTRY结构
        pLdrDataEntry =
(PLDR_DATA_TABLE_ENTRY)CONTAINING_RECORD(pListEntryStart,LDR_DATA_TABLE_ENTRY,InMemoryOrderLinks);

        //5、输出_LDR_DATA_TABLE_ENTRY的BaseDllName或FullDllName成员信息
        printf("%S/n",pLdrDataEntry->BaseDllName.Buffer);

        pListEntryStart = pListEntryStart->Flink;

    }while(pListEntryStart != pListEntryEnd);
}

/*
output:
EnumInLoadModule.exe
ntdll.dll
kernel32.dll
ResLibDemo.dll
...
```

```
*/


typedef struct _LDR_DATA_TABLE_ENTRY
{

    LIST_ENTRY InLoadOrderLinks;
    LIST_ENTRY InMemoryOrderLinks;
    LIST_ENTRY InInitializationOrderLinks;
    PVOID DllBase;
    PVOID EntryPoint;
    ULONG SizeOfImage;
    UNICODE_STRING FullDllName;
    UNICODE_STRING BaseDllName;
    ULONG Flags;
    USHORT LoadCount;
    USHORT TlsIndex;
    union
    {
        LIST_ENTRY HashLinks;
        struct
        {
            PVOID SectionPointer;
            ULONG CheckSum;
        };
    };
    ULONG TimeDateStamp;

} LDR_DATA_TABLE_ENTRY, *PLDR_DATA_TABLE_ENTRY;


///

 xp下
```

```
0:000> dt _peb 7ffde000
ntdll!_PEB
   +0x000 InheritedAddressSpace : 0 ''
   +0x001 ReadImageFileExecOptions : 0 ''
   +0x002 BeingDebugged    : 0x1 ''
   +0x003 SpareBool        : 0 ''
   +0x004 Mutant           : 0xffffffff
   +0x008 ImageBaseAddress : 0x00400000
   +0x00c Ldr              : 0x00251ea0 _PEB_LDR_DATA
   +0x010 ProcessParameters : 0x00020000 _RTL_USER_PROCESS_PARAMETERS
   +0x014 SubSystemData    : (null)
   +0x018 ProcessHeap      : 0x00150000
   +0x01c FastPebLock      : 0x7c9a0620 _RTL_CRITICAL_SECTION
   +0x020 FastPebLockRoutine : 0x7c921000
   +0x024 FastPebUnlockRoutine : 0x7c9210e0
   +0x028 EnvironmentUpdateCount : 1
   //.............

0:000> dt _PEB_LDR_DATA 0x00251ea0
ntdll!_PEB_LDR_DATA
   +0x000 Length           : 0x28
   +0x004 Initialized      : 0x1 ''
   +0x008 SsHandle         : (null)
   +0x00c InLoadOrderModuleList : _LIST_ENTRY [ 0x251ee0 - 0x252a80 ]            //下面用到，指向一
个 _LDR_DATA_TABLE_ENTRY的首地址
   +0x014 InMemoryOrderModuleList : _LIST_ENTRY [ 0x251ee8 - 0x252a88 ]      //下面用到，指向一个
_LDR_DATA_TABLE_ENTRY的InMemoryOrderLinks地址
   +0x01c InInitializationOrderModuleList : _LIST_ENTRY [ 0x251f58 - 0x2529d0 ]  //下面用到，指向一个
_LDR_DATA_TABLE_ENTRY的InInitializationOrderLinks地址
   +0x024 EntryInProgress  : (null)
0:000> dt _LDR_DATA_TABLE_ENTRY 0x251ee0
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x251f48 - 0x251eac ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x251f50 - 0x251eb4 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x0 - 0x0 ]
   +0x018 DllBase          : 0x00400000
   +0x01c EntryPoint       : 0x00415667
   +0x020 SizeOfImage      : 0x88000
   +0x024 FullDllName      : _UNICODE_STRING "C:\Documents and Settings\Administrator\桌面\Dbgview.exe"
   +0x02c BaseDllName      : _UNICODE_STRING "Dbgview.exe"
   +0x034 Flags            : 0x5000
   +0x038 LoadCount        : 0xffff
   +0x03a TlsIndex         : 0
   +0x03c HashLinks        : _LIST_ENTRY [ 0x7c99e298 - 0x7c99e298 ]
   +0x03c SectionPointer   : 0x7c99e298
   +0x040 CheckSum         : 0x7c99e298
   +0x044 TimeDateStamp    : 0x4ddfd5aa
   +0x044 LoadedImports    : 0x4ddfd5aa
   +0x048 EntryPointActivationContext : (null)
   +0x04c PatchInformation : (null)
0:000> dt   LDR  DATA  TABLE  ENTRY 0x251f48
```

ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [0x252010 - 0x251ee0]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [0x252018 - 0x251ee8]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [0x252020 - 0x251ebc]
   +0x018 DllBase   : 0x7c920000
   +0x01c EntryPoint   : 0x7c9320f8
   +0x020 SizeOfImage   : 0x96000
   +0x024 FullDllName   : _UNICODE_STRING "C:\WINDOWS\system32\ntdll.dll"
   +0x02c BaseDllName   : _UNICODE_STRING "ntdll.dll"
   +0x034 Flags   : 0x85004
   +0x038 LoadCount   : 0xffff
   +0x03a TlsIndex   : 0
   +0x03c HashLinks   : _LIST_ENTRY [0x7c99e2e8 - 0x7c99e2e8]
   +0x03c SectionPointer   : 0x7c99e2e8
   +0x040 CheckSum   : 0x7c99e2e8
   +0x044 TimeDateStamp   : 0x4d00f280
   +0x044 LoadedImports   : 0x4d00f280
   +0x048 EntryPointActivationContext : (null)
   +0x04c PatchInformation : (null)
0:000> dt _LDR_DATA_TABLE_ENTRY 0x251ee8-0x8 //(计算根据偏移得到首地址)
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [0x251f48 - 0x251eac]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [0x251f50 - 0x251eb4]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [0x0 - 0x0]
   +0x018 DllBase   : 0x00400000
   +0x01c EntryPoint   : 0x00415667
   +0x020 SizeOfImage   : 0x88000
   +0x024 FullDllName   : _UNICODE_STRING "C:\Documents and Settings\Administrator\桌面\Dbgview.exe"
   +0x02c BaseDllName   : _UNICODE_STRING "Dbgview.exe"
   +0x034 Flags   : 0x5000
   +0x038 LoadCount   : 0xffff
   +0x03a TlsIndex   : 0
   +0x03c HashLinks   : _LIST_ENTRY [0x7c99e298 - 0x7c99e298]
   +0x03c SectionPointer   : 0x7c99e298
   +0x040 CheckSum   : 0x7c99e298
   +0x044 TimeDateStamp   : 0x4ddfd5aa
   +0x044 LoadedImports   : 0x4ddfd5aa
   +0x048 EntryPointActivationContext : (null)
   +0x04c PatchInformation : (null)

0:000> dt _LDR_DATA_TABLE_ENTRY 0x251f50-0x8 //(计算根据偏移得到首地址)
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [0x252010 - 0x251ee0]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [0x252018 - 0x251ee8]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [0x252020 - 0x251ebc]
   +0x018 DllBase   : 0x7c920000
   +0x01c EntryPoint   : 0x7c9320f8
   +0x020 SizeOfImage   : 0x96000
   +0x024 FullDllName   : _UNICODE_STRING "C:\WINDOWS\system32\ntdll.dll"
   +0x02c BaseDllName   : _UNICODE_STRING "ntdll.dll"

```
   +0x034 Flags            : 0x85004
   +0x038 LoadCount        : 0xffff
   +0x03a TlsIndex         : 0
   +0x03c HashLinks        : _LIST_ENTRY [ 0x7c99e2e8 - 0x7c99e2e8 ]
   +0x03c SectionPointer   : 0x7c99e2e8
   +0x040 CheckSum         : 0x7c99e2e8
   +0x044 TimeDateStamp    : 0x4d00f280
   +0x044 LoadedImports    : 0x4d00f280
   +0x048 EntryPointActivationContext : (null)
   +0x04c PatchInformation : (null)
0:000> dt _LDR_DATA_TABLE_ENTRY 0x251f58-0x10
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x252010 - 0x251ee0 ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x252018 - 0x251ee8 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x252020 - 0x251ebc ]
   +0x018 DllBase          : 0x7c920000
   +0x01c EntryPoint       : 0x7c9320f8
   +0x020 SizeOfImage      : 0x96000
   +0x024 FullDllName      : _UNICODE_STRING "C:\WINDOWS\system32\ntdll.dll"
   +0x02c BaseDllName      : _UNICODE_STRING "ntdll.dll"
   +0x034 Flags            : 0x85004
   +0x038 LoadCount        : 0xffff
   +0x03a TlsIndex         : 0
   +0x03c HashLinks        : _LIST_ENTRY [ 0x7c99e2e8 - 0x7c99e2e8 ]
   +0x03c SectionPointer   : 0x7c99e2e8
   +0x040 CheckSum         : 0x7c99e2e8
   +0x044 TimeDateStamp    : 0x4d00f280
   +0x044 LoadedImports    : 0x4d00f280
   +0x048 EntryPointActivationContext : (null)
   +0x04c PatchInformation : (null)
0:000> dt _LDR_DATA_TABLE_ENTRY 0x252020-0x10
ntdll!_LDR_DATA_TABLE_ENTRY
   +0x000 InLoadOrderLinks : _LIST_ENTRY [ 0x2520d0 - 0x251f48 ]
   +0x008 InMemoryOrderLinks : _LIST_ENTRY [ 0x2520d8 - 0x251f50 ]
   +0x010 InInitializationOrderLinks : _LIST_ENTRY [ 0x252328 - 0x251f58 ]
   +0x018 DllBase          : 0x7c800000
   +0x01c EntryPoint       : 0x7c80b64e
   +0x020 SizeOfImage      : 0x11e000
   +0x024 FullDllName      : _UNICODE_STRING "C:\WINDOWS\system32\kernel32.dll"
   +0x02c BaseDllName      : _UNICODE_STRING "kernel32.dll"
   +0x034 Flags            : 0x85004
   +0x038 LoadCount        : 0xffff
   +0x03a TlsIndex         : 0
   +0x03c HashLinks        : _LIST_ENTRY [ 0x7c99e2d0 - 0x7c99e2d0 ]
   +0x03c SectionPointer   : 0x7c99e2d0
   +0x040 CheckSum         : 0x7c99e2d0
   +0x044 TimeDateStamp    : 0x506bc5dd
   +0x044 LoadedImports    : 0x506bc5dd
   +0x048 EntryPointActivationContext : (null)
   +0x04c PatchInformation : (null)
```

///

win7下 上面windbg 手工方式一样，只是运行结果略有一同，下面给出程序 实现的方式。

```
VOID GetPebModule()
{
 PBYTE pPeb;
 PPEB_LDR_DATA pLdr , tttt;
 PLDR_DATA_TABLE_ENTRY pLdrData;

 __asm
 {
  MOV EAX, FS:[0x18]
  MOV EAX, [EAX + 0x30]

  MOV pPeb, EAX
 }

 pLdr = (PPEB_LDR_DATA)(*(DWORD*)(pPeb + 0x0C));

 tttt = pLdr;

 printf("\nInLoadOrderModuleList:\n");

 pLdrData = (PLDR_DATA_TABLE_ENTRY)pLdr->InLoadOrderModuleList.Flink;

 for ( ; (pLdr->InLoadOrderModuleList.Flink) != (pLdrData->InLoadOrderLinks.Flink); )
 {
  wprintf(L"%s\r\n", (pLdrData->FullDllName.Buffer));
  wprintf(L"%s\r\n", (pLdrData->BaseDllName.Buffer));
  pLdrData = (PLDR_DATA_TABLE_ENTRY)(pLdrData->InLoadOrderLinks.Flink);
 }


 printf("\n\nInInitializationOrderModuleList:\n");

 pLdr = tttt;

 pLdrData = (PLDR_DATA_TABLE_ENTRY)CONTAINING__RECORD(tttt->InInitializationOrderModuleList.Flink, LDR_DATA

 for ( ; (tttt->InInitializationOrderModuleList.Flink) != (pLdrData->InInitializationOrderLinks.Flink); )
 {
  wprintf(L"%s\r\n", (pLdrData->FullDllName.Buffer));
  wprintf(L"%s\r\n", (pLdrData->BaseDllName.Buffer));
  pLdrData = (PLDR_DATA_TABLE_ENTRY)CONTAINING__RECORD(pLdrData->InInitializationOrderLinks.Flink, LDR_DATA

 }

 printf("\n\nInMemoryOrderModuleList:\n");
 pLdrData = (PLDR_DATA_TABLE_ENTRY)CONTAINING__RECORD(pLdr->InMemoryOrderModuleList.Flink, LDR_DATA_TABLE_E

 for ( ; (pLdr->InMemoryOrderModuleList.Flink) != (pLdrData->InMemoryOrderLinks.Flink); )
 {
  wprintf(L"%s\r\n", (pLdrData->FullDllName.Buffer));
  wprintf(L"%s\r\n", (pLdrData->BaseDllName.Buffer));
  pLdrData = (PLDR_DATA_TABLE_ENTRY)CONTAINING__RECORD(pLdrData->InMemoryOrderLinks.Flink, LDR_DATA_TABLE_E

 }

 return ;
}
```

/*

输出：

nInLoadOrderModuleList:

C:\Windows\SYSTEM32\ntdll.dll
ntdll.dll
C:\Windows\system32\kernel32.dll
kernel32.dll
C:\Windows\system32\KERNELBASE.dll
KERNELBASE.dll

InInitializationOrderModuleList:
C:\Windows\SYSTEM32\ntdll.dll
ntdll.dll
C:\Windows\system32\KERNELBASE.dll
KERNELBASE.dll
C:\Windows\system32\kernel32.dll
kernel32.dll

InMemoryOrderModuleList:
d:\Desktop\HsdaosUI\Hsdaos\trunk\Bin\Vstest.exe
Vstest.exe
C:\Windows\SYSTEM32\ntdll.dll
ntdll.dll
C:\Windows\system32\kernel32.dll
kernel32.dll
C:\Windows\system32\KERNELBASE.dll
KERNELBASE.dll

*/

///

用shellcode获取Kernel32基地址的几种方法

win7下

```
__asm
{
 push    eax
 mov     eax, fs:[30h]  //PEB 址，mov  eax, fs:[18h] 这个获得TEB地址
 mov     eax, [eax + 0Ch]
 mov     eax, [eax + 1Ch]
 mov     eax, [eax]
 mov     eax, [eax]
 mov     eax, [eax + 8]
 mov   hModule, eax
 pop     eax
}
```

mov eax, fs:[0] //为_EXCEPTION_REGISTRATION_RECORD的地址，exceptionList

```
__asm
{
 push    eax
 mov     eax, fs:[18h] //TEB地址

 mov     eax, [eax + 30h ] //PEB地址
 mov     eax, [eax + 0Ch]
 mov     eax, [eax + 1Ch]
 mov     eax, [eax]
 mov     eax, [eax]
 mov     eax, [eax + 8]
 mov   hModule, eax
 pop     eax
}
```

XP 下

```
__asm
{
 push    eax
 mov     eax, fs:[30h]
 mov     eax, [eax + 0Ch]
 mov     eax, [eax + 1Ch]
 mov     eax, [eax]
 mov     eax, [eax + 8]
 mov   hModule, eax
 pop     eax
}
```

1、CreateProcess函数在完成装载应用程序后，会先将一个返回地址压入到堆栈顶端，而这个返回地址恰好在Kernel32.dll中，利用这个原理我们可以顺着这个返回地址按64KB大小往地址搜索，那么我们一定可以找到Kernel32模块的基地址，废话少说，代码如下：

```
GetK32Base:
   mov eax, [esp+04h]    ;得到kernel32的返回地址
   and eax, ffff0000h
Search:
   cmp WORD ptr [eax], IMAGE_DOS_SIGNATURE   ;判断是否为ImageBase
   jnz @f
   mov edx, [eax+3ch]
   add edx, eax
   cmp WORD ptr [edx], IMAGE_NT_SIGNATURE   ;判断是否为PE头
   jnz @f
   ret
@@:
   dec eax    ;按64KB递减搜索
   xor ax, ax
   cmp eax, 70000000h
   ja Search
   ret
```

2、通过PEB枚举当前进程空间中用户模块列表也可以获取Kernel32模块的基地址，fs:[0]指向TEB，fs:[30h]指向PEB，PEB偏移0ch是LDR指针，以下可以分别通过加载顺序、内存顺序、初始化顺序获取Kernel32模块的基地址，这里以初始化顺序为例：

未公开的LDR_MODULE数据结构如下：

```
typedef struct _LDR_MODULE
{
   LIST_ENTRY      InLoadOrderModuleList;          // +0x00
   LIST_ENTRY      InMemoryOrderModuleList;        // +0x08
   LIST_ENTRY      InInitializationOrderModuleList; // +0x10
   PVOID           BaseAddress;                    // +0x18
   PVOID           EntryPoint;                     // +0x1c
   ULONG           SizeOfImage;                    // +0x20
   UNICODE_STRING  FullDllName;                    // +0x24
   UNICODE_STRING  BaseDllName;                    // +0x2c
   ULONG           Flags;                          // +0x34
   SHORT           LoadCount;                      // +0x38
   SHORT           TlsIndex;                       // +0x3a
   LIST_ENTRY      HashTableEntry;                 // +0x3c
   ULONG           TimeDateStamp;                  // +0x44
                                                   // +0x48
} LDR_MODULE, *PLDR_MODULE;
```

以下是WinDbg显示的PEB_LDR_DATA的数据结构

```
+0x00c Ldr              : Ptr32 to struct _PEB_LDR_DATA, 7 elements, 0x28 bytes
   +0x000 Length        : Uint4B
   +0x004 Initialized   : UChar
   +0x008 SsHandle      : Ptr32 to Void
   +0x00c InLoadOrderModuleList : struct _LIST_ENTRY, 2 elements, 0x8 bytes
      +0x000 Flink      : Ptr32 to struct _LIST_ENTRY, 2 elements, 0x8 bytes
      +0x004 Blink      : Ptr32 to struct _LIST_ENTRY, 2 elements, 0x8 bytes
   +0x014 InMemoryOrderModuleList : struct _LIST_ENTRY, 2 elements, 0x8 bytes
      +0x000 Flink      : Ptr32 to struct _LIST_ENTRY, 2 elements, 0x8 bytes
      +0x004 Blink      : Ptr32 to struct _LIST_ENTRY, 2 elements, 0x8 bytes
   +0x01c InInitializationOrderModuleList : struct _LIST_ENTRY, 2 elements, 0x8 bytes
      +0x000 Flink      : Ptr32 to struct _LIST_ENTRY, 2 elements, 0x8 bytes
      +0x004 Blink      : Ptr32 to struct _LIST_ENTRY, 2 elements, 0x8 bytes
   +0x024 EntryInProgress : Ptr32 to Void
```

通过上面两个结构我们可以通过3种方式获取Kernel32模块的基地址

1)加载顺序的方式：

```
GetK32Base：
assume fs:nothing
push esi
xor eax,eax
mov eax,fs:[eax+30h]    ;指向PEB的指针
mov eax,[eax+0ch]    ;指向PEB_LDR_DATA的指针
mov esi,[eax+0ch]    ;根据PEB_LDR_DATA得出InLoadOrderModuleList的Flink字段
lodsd
mov eax, [eax]     ;指向下一个节点
mov eax,[eax+18h]    ;Kernel.dll的基地址
pop esi
ret
```

2)内存顺序的方式：

```
GetK32Base：
assume fs:nothing
push esi
xor eax,eax
mov eax,fs:[eax+30h]    ;指向PEB的指针
mov eax,[eax+0ch]    ;指向PEB_LDR_DATA的指针
mov esi,[eax+14h]    ;根据PEB_LDR_DATA得出InMemoryOrderModuleList的Flink字段
lodsd
mov eax, [eax]     ;指向下一个节点
mov eax,[eax+10h]    ;Kernel.dll的基地址
pop esi
ret
```

3)初始化顺序的方式：

GetK32Base：

```
assume fs:nothing
push esi
xor eax,eax
mov eax,fs:[eax+30h]    ;指向PEB的指针
mov eax,[eax+0ch]    ;指向PEB_LDR_DATA的指针
mov esi,[eax+1ch]    ;根据PEB_LDR_DATA得出InInitializationOrderModuleList的Flink字段
lodsd
mov eax,[eax+08h]    ;Kernel.dll的基地址
pop esi
ret
```

3、通过遍历SEH链的方法，在SEH链中查找成员prev的值为0xFFFFFFFFh的EXCEPTION_REGISTER结构。该结构中的handler值是系统异常处理例程，总是位于kernerl32.dll中。当前线程的TIB保存在fs段选择器指定的数据段的0偏移处，所以fs:[0] 的地方就是TIB结构中的ExceptionList字段。而ExceptionList指向一个EXCEPTION_REGISTRATION结构，SEH异常处理回调函数的入口地址就是由EXCEPTION_REGISTRATION结构指定。

TIB结构如下：

```
NT_TIB STRUCT
ExceptionList dd   ?
StackBase dd     ?
StackLimit dd     ?
SubSystemTib dd   ?
union
  FiberData dd   ?
  Version dd     ?
ends
ArbitraryUserPointer dd ?
Self dd ?
NT_TIB ENDS

EXCEPTION_REGISTRATION STRUCT
prev      dd      ?      ;前一个EXCEPTION_REGISTRATION结构的地址
handler   dd      ?      ;异常处理回调函数地址
EXCEPTION_REGISTRATION      ends
```

GetK32Base：

```
assume fs:nothing
xor esi, esi
mov esi, fs:[esi]
lodsd
@@:
inc eax
jz @f
dec eax
xchg esi, eax
lodsd
jmp @b
@@:
lodsd
@@:
dec eax
xor ax, ax
cmp DWORD ptr [eax], 5A4D
jnz @b
ret
```

结束语：技术可以做正义的事，也可以做邪恶的事，关键是看什么人掌握这些技术