TEB & PEB (Thread and Process Environment Block)

---

1. Introducion

I read many times about PEB and TEB and ofcourse about using those „things” to locate kernel32.dll. I think everyone saw this code fragment or something similar:

```
mov eax,fs[030h]
mov eax,[eax+0ch]
mov eax,[eax+0ch]
mov eax,[eax]
mov eax,[eax+08h]
```

Okay, it works... but how? Why? I always wanted to know what exactly is TEB and PEB, how it looks like and how I can use it. I tried to find something in google but there weren't many interesting results. Documentation on msdn is not complete, many fields are „reserved” and there is always information: [This structure may be altered in future versions of Windows.] .

So I decided to figure it out by myself – maybe there are some materials about this subject on www but I didn't find any so I also decided to write this small article for anyone who's interested. Sory for my bad english :). If you will find any errors/mistakes just mail me **alek.barszczewski@gmail.com**.

---

2. Getting TEB & PEB structure

First of all I had to get the structures of TEB & PEB on my system (vista 32bit sp2). I downloaded and installed free MC Debugging Tools for Windows and Windows Symbols from the same website. I started WinDbg and added symbol path (File->Symbol File Path). Then I opened some executable and the command line appeared. There are some very useful commands:

```
dt ntdll!_peb                 // it shows us structure of peb
dt ntdll!_teb                 // it shows us structure of teb
dt ntdll!_peb_ldr_data        // it shows us structure of loader data
dt ntdll!<StructureName> // it shows us structure of any structure

dt ntdll!_peb @$peb         // it shows peb structure of current process
```

```
dt ntdll!_peb @$teb        // it shows peb structure of current process
!peb                       // it shows peb for current process
!teb                       // it shows teb for current process
dd <address> L<n>          // it dumps <n> dwords from <address>
                           // for example dd $peb L2 will dump 2 dwords
                           // from address of peb
```

And for example here is dump of _TEB structure:

```
0:000> dt ntdll!_teb
   +0x000 NtTib            : _NT_TIB
   +0x01c EnvironmentPointer : Ptr32 Void
   +0x020 ClientId         : _CLIENT_ID
   +0x028 ActiveRpcHandle  : Ptr32 Void
   +0x02c ThreadLocalStoragePointer : Ptr32 Void
   +0x030 ProcessEnvironmentBlock : Ptr32 _PEB
   +0x034 LastErrorValue   : Uint4B
   +0x038 CountOfOwnedCriticalSections : Uint4B
   +0x03c CsrClientThread  : Ptr32 Void
   +0x040 Win32ThreadInfo  : Ptr32 Void
   +0x044 User32Reserved   : [26] Uint4B
   +0x0ac UserReserved     : [5] Uint4B
   +0x0c0 WOW32Reserved    : Ptr32 Void
   +0x0c4 CurrentLocale    : Uint4B
   +0x0c8 FpSoftwareStatusRegister : Uint4B
   +0x0cc SystemReserved1  : [54] Ptr32 Void
   +0x1a4 ExceptionCode    : Int4B
   +0x1a8 ActivationContextStackPointer : Ptr32 _ACTIVATION_CONTEXT_STACK
   +0x1ac SpareBytes1      : [36] UChar
   +0x1d0 TxFsContext      : Uint4B
   +0x1d4 GdiTebBatch      : _GDI_TEB_BATCH
   +0x6b4 RealClientId     : _CLIENT_ID
   +0x6bc GdiCachedProcessHandle : Ptr32 Void
   +0x6c0 GdiClientPID     : Uint4B
   +0x6c4 GdiClientTID     : Uint4B
   +0x6c8 GdiThreadLocalInfo : Ptr32 Void
   +0x6cc Win32ClientInfo  : [62] Uint4B
   +0x7c4 glDispatchTable  : [233] Ptr32 Void
   +0xb68 glReserved1      : [29] Uint4B
   +0xbdc glReserved2      : Ptr32 Void
   +0xbe0 glSectionInfo    : Ptr32 Void
   +0xbe4 glSection        : Ptr32 Void
   +0xbe8 glTable          : Ptr32 Void
   +0xbec glCurrentRC      : Ptr32 Void
   +0xbf0 glContext        : Ptr32 Void
   +0xbf4 LastStatusValue  : Uint4B
   +0xbf8 StaticUnicodeString : _UNICODE_STRING
   +0xc00 StaticUnicodeBuffer : [261] Wchar
   +0xe0c DeallocationStack : Ptr32 Void
   +0xe10 TlsSlots         : [64] Ptr32 Void
   +0xf10 TlsLinks         : _LIST_ENTRY
   +0xf18 Vdm              : Ptr32 Void
```

+0xf1c ReservedForNtRpc : Ptr32 Void
+0xf20 DbgSsReserved : [2] Ptr32 Void
+0xf28 HardErrorMode : Uint4B
+0xf2c Instrumentation : [9] Ptr32 Void
+0xf50 ActivityId : _GUID
+0xf60 SubProcessTag : Ptr32 Void
+0xf64 EtwLocalData : Ptr32 Void
+0xf68 EtwTraceData : Ptr32 Void
+0xf6c WinSockData : Ptr32 Void
+0xf70 GdiBatchCount : Uint4B
+0xf74 SpareBool0 : UChar
+0xf75 SpareBool1 : UChar
+0xf76 SpareBool2 : UChar
+0xf77 IdealProcessor : UChar
+0xf78 GuaranteedStackBytes : Uint4B
+0xf7c ReservedForPerf : Ptr32 Void
+0xf80 ReservedForOle : Ptr32 Void
+0xf84 WaitingOnLoaderLock : Uint4B
+0xf88 SavedPriorityState : Ptr32 Void
+0xf8c SoftPatchPtr1 : Uint4B
+0xf90 ThreadPoolData : Ptr32 Void
+0xf94 TlsExpansionSlots : Ptr32 Ptr32 Void
+0xf98 ImpersonationLocale : Uint4B
+0xf9c IsImpersonating : Uint4B
+0xfa0 NlsCache : Ptr32 Void
+0xfa4 pShimData : Ptr32 Void
+0xfa8 HeapVirtualAffinity : Uint4B
+0xfac CurrentTransactionHandle : Ptr32 Void
+0xfb0 ActiveFrame : Ptr32 _TEB_ACTIVE_FRAME
+0xfb4 FlsData : Ptr32 Void
+0xfb8 PreferredLanguages : Ptr32 Void
+0xfbc UserPrefLanguages : Ptr32 Void
+0xfc0 MergedPrefLanguages : Ptr32 Void
+0xfc4 MuiImpersonation : Uint4B
+0xfc8 CrossTebFlags : Uint2B
+0xfc8 SpareCrossTebBits : Pos 0, 16 Bits
+0xfca SameTebFlags : Uint2B
+0xfca DbgSafeThunkCall : Pos 0, 1 Bit
+0xfca DbgInDebugPrint : Pos 1, 1 Bit
+0xfca DbgHasFiberData : Pos 2, 1 Bit
+0xfca DbgSkipThreadAttach : Pos 3, 1 Bit
+0xfca DbgWerInShipAssertCode : Pos 4, 1 Bit
+0xfca DbgRanProcessInit : Pos 5, 1 Bit
+0xfca DbgClonedThread : Pos 6, 1 Bit
+0xfca DbgSuppressDebugMsg : Pos 7, 1 Bit
+0xfca RtlDisableUserStackWalk : Pos 8, 1 Bit
+0xfca RtlExceptionAttached : Pos 9, 1 Bit
+0xfca SpareSameTebBits : Pos 10, 6 Bits
+0xfcc TxnScopeEnterCallback : Ptr32 Void
+0xfd0 TxnScopeExitCallback : Ptr32 Void
+0xfd4 TxnScopeContext : Ptr32 Void
+0xfd8 LockCount : Uint4B

```
+0xfdc ProcessRundown   : Uint4B
+0xfe0 LastSwitchTime   : Uint8B
+0xfe8 TotalSwitchOutTime : Uint8B
+0xff0 WaitReasonBitMap : _LARGE_INTEGER
```

---

3. TEB (Thread Environment Block)

Okay but what is TEB and where it is? Each process and each thread must be represented by some structure in the system. In kernel mode thread is represented by structures ETHREAD (executive thread) and KTHREAD (kernel thread). But windows loader and other dll's in user mode need some data about thread (and process) to be accessible in user mode. And here comes TEB (and also PEB, explained in next chapter). TEB is just some structure which contains thread data which is accessible in user mode. But where is it? I thought that in protected mode and flat memory model I wont use any segment anymore – I wasn't right :). When code in any thread starts to execute the segment FS „points" or „maps" to TEB. So the instruction:

```
mov eax,fs:[012h]
```

means „move to eax a dword from TEB+offset 12h". We can also get direct address of TEB – as you can see in _TEB structure it starts with structure _NT_TIB which is called Thread Information Block and as I know it's system version independent. Here is structure of _NT_TIB:

```
_NT_TIB struct

   ExceptionList          dword ?  ; Ptr32 _EXCEPTION_REGISTRATION_RECORD
   StackBase              dword ?  ; Ptr32 Void
   StackLimit             dword ?  ; Ptr32 Void
   SubSystemTib           dword ?  ; Ptr32 Void
   union
      FiberData           dword ?  ; Ptr32 Void
      Version             dword ?  ; Uint4B
   ends
   ArbitraryUserPointer   dword ?  ; Ptr32 Void
   Self                   dword ?  ; Ptr32 _NT_TIB

_NT_TIB ends
```

I made all structures (_PEB, _TEB, _NT_TIB and many others) as masm .inc files – you can download them from this link. I wrote those structures just using WinDbg so they are correct for my windows vista sp2 and I am not sure how they look like on for example win2k or xp – but as I know they are very similar and only few fields maybe different. You can try those structures, they will probably work on other systems (I don't realy know about win9x but I am sure it wont work there as win9x is not NT system). However I didn't test every struct I wrote so there may be some mistakes (like dword instead of word) but most of them works correctly.

But lets go back to the subject – how to get linear address of _TEB structure? In _NT_TIB structure there is a field Self and it's a pointer (address) to _NT_TIB structure – to itself. As the _NT_TIB structure is at the begining of _TEB structure so this address is also address of TEB :). So:

```
assume fs:nothing
mov eax,fs:[018h]
```

loads eax with address of _TEB structure. Using the files I wrote we can write like this:

```
assume fs:nothing
mov eax,fs:[_TEB.NtTib.Self]
```

and it will be the same. Then we can write

```
assume eax:ptr _TEB
mov ebx,[eax].<any _teb member> ; for example mov ebx,[eax].ActiveRpcHandle
```

Okay, I wont describe every member of _TEB structure because most of them I have no idea what they are :), but I don't think they could be usefull. But there are few interesting:

_TEB.NtTib.StackBase – base address of stack begining
_TEB.NtTib.StackLimit - address of the top of the stack

and the most interesting:

_TEB.ProcessEnvironmentBlock – pointer to _PEB :). It is on offset 030h. So we can get it writing for example:

```
assume fs:nothing
mov eax,fs:[030h] ; do you remember this line from the code block at the begining?
```

or

```
mov eax,fs:[_TEB.ProcessEnvironmentBlock]
```

Now as we have address of _PEB structure we will jump to next chapter.

---

4. PEB (Process Environment Block)

_PEB is large struct so I wont put it here – you can just look into _PEB.inc file. PEB as TEB is just a user mode accessible information block about a process (while in kernel mode process is represented by EPROCESS and KPROCESS). There are many interesting members in PEB structure. For example

_PEB.ImageBaseAddress
_PEB.ProcessParameters
_PEB.OSMajorVersion
_PEB.OSMinorVersion
_PEB.OSBuildNumber

and many others. I think they are self-explanatory.
There are 2 members that I want to focus on as they are very important. First is _PEB.BeingDebugged byte. This filed is 1 if there is debugger present. It's used in simplest anti-debugging technique. Now we can write some piece of code which detects debugger:

```
assume fs:nothing
mov eax,fs:[030h]  ; eax = _PEB
mov al,[eax+02h]   ; al = BeingDebugged
cmp al,01h         ; check if it's set to 1
je @debugger_found

@debugger_not_found:
  ....
@debugger_found:
  ....
```

The second field is _PEB.Ldr which is a dword pointer to _PEB_LDR_DATA structure which will be explained in next chapter.

---

5. _PEB_LDR_DATA (Loader Data)

This structure is as follows:

```
_PEB_LDR_DATA struct

  Length_                     dword ?       ; Uint4B
  Initialized                 dword ?       ; UChar
  SsHandle                    dword ?       ; Ptr32 Void
  InLoadOrderModuleList       _LIST_ENTRY <> ; (~ptr _LDR_DATA_TABLE_ENTRY)
  InMemoryOrderModuleList     _LIST_ENTRY <> ; (~ptr _LDR_DATA_TABLE_ENTRY)
  InInitializationOrderModuleList _LIST_ENTRY <> ;  (~ptr _LDR_DATA_TABLE_ENTRY)
  EntryInProgress             dword ?       ; Ptr32 Void
  ShutdownInProgress          dword ?       ; UChar
  ShutdownThreadId            dword ?       ; Ptr32 Void

_PEB_LDR_DATA ends
```

This structure contains data used by windows loader. We have Initialized flag and ShutdownInProgress flag and so on. But the most interesting for us are 3 list entries.

```
InLoadOrderModuleList
InMemoryOrderModuleList
InInitializationOrderModuleList
```

Those are entries to linked lists of _LDR_DATA_TABLE_ENTRY structures. Each of such TABLE_ENTRY structure contains information about loaded module used by the process. For example executable file that do nothing, I mean it just for example invoke Exitprocess,1, will have loaded 3 modules – itself (program.exe), kernel32.dll, and ntdll.dll. Kernel32.dll and ntdll.dll are required for every process (I think so :). (It's on windows vista sp2 – for example on windows 7 there is one extra dll loaded called as I remember ntoskrnl.dll or something like that). But to obtain informations about modules we have to know how those lists are built.

```
_LIST_ENTRY struct

  Flink dword ?  ; Ptr32 _LIST_ENTRY
  Blink dword ?  ; Ptr32 _LIST_ENTRY
```

```
_LIST_ENTRY ends
```

Each list entry has 2 pointers – first points forward and the second backward (it's double linked list). So lets take InLoadOrderModuleList.flink. Where it points? To the LDR_DATA_TABLE_ENTRY:

```
_LDR_DATA_TABLE_ENTRY struct

   InLoadOrderLinks            _LIST_ENTRY <>     ; _LIST_ENTRY
   InMemoryOrderLinks          _LIST_ENTRY <>     ; _LIST_ENTRY
   InInitializationOrderLinks  _LIST_ENTRY <>     ; _LIST_ENTRY

   DllBase                     dword ?            ; Ptr32 Void
   EntryPoint                  dword ?            ; Ptr32 Void
   SizeOfImage                 dword ?            ; Uint4B
   FullDllName                 _UNICODE_STRING <>  ; _UNICODE_STRING
   BaseDllName                 _UNICODE_STRING <>  ; _UNICODE_STRING
   Flags                       dword ?            ; Uint4B
   LoadCount                   word  ?            ; Uint2B
   TlsIndex                    word  ?            ; Uint2B
   union
      HashLinks                _LIST_ENTRY <>     ; _LIST_ENTRY
      SectionPointer           dword ?            ; Ptr32 Void
   ends
   CheckSum                    dword ?            ; Uint4B
   union
     TimeDateStamp             dword ?            ; Uint4B
     LoadedImports             dword ?            ; Ptr32 Void
   ends
   EntryPointActivationContext dword ?            ; Ptr32 _ACTIVATION_CONTEXT (UnDocumented)
   PatchInformation            dword ?            ; Ptr32 Void
   ForwarderLinks              _LIST_ENTRY <>     ; _LIST_ENTRY
   ServiceTagLinks             _LIST_ENTRY <>     ; _LIST_ENTRY
   StaticLinks                 _LIST_ENTRY <>     ; _LIST_ENTRY

_LDR_DATA_TABLE_ENTRY ends
```

Each of those entries has 3 fields:

```
InLoadOrderLinks            _LIST_ENTRY <>     ; _LIST_ENTRY
InMemoryOrderLinks          _LIST_ENTRY <>     ; _LIST_ENTRY
InInitializationOrderLinks  _LIST_ENTRY <>     ; _LIST_ENTRY
```

I expected that it will have only one _LIST_ENTRY structure which will have pointer to the next _LDR_DATA_TABLE_ENTRY. But it has 3 entries. It's because each module used by a process has ONLY one _LDR_DATA_TABLE_ENTRY but this structure is used by 3 different linked lists. I mean that each list (InLoadOrderModuleList and InMemoryOrderModuleList and InInitializationOrderModuleList) contains SAME _LDR_DATA_TABLE_ENTRY structures but in different order. I causes some problems with addressing those structures from list entries. For example if we will take flink from InLoadOrderModuleList it will point to the begining of _LDR_DATA_TABLE_ENTRY. But when we will take flink from InMemoryOrderModuleList it wont point to the begining but to the InMemoryOrderLinks field in _LDR_DATA_TABLE_ENTRY. It means if we want to get pointer to begining of _LDR_DATA_TABLE_ENTRY we have to substitute 0x8h (sizeof

_LIST_ENTRY structure) from flink from InMemoryOrderModuleList. For InInitializationOrderModuleList we have to sub 0x10h respectively.

---

7. _LDR_DATA_TABLE_ENTRY

Now lets focus on _LDR_DATA_TABLE_ENTRY structure. Firste few fileds after 3 list entries are:

```
DllBase                 dword ?                 ; Ptr32 Void
EntryPoint              dword ?                 ; Ptr32 Void
SizeOfImage             dword ?                 ; Uint4B
FullDllName             _UNICODE_STRING <>  ; _UNICODE_STRING
BaseDllName             _UNICODE_STRING <>  ; _UNICODE_STRING
```

Interesting huh? We can check dll name and then we have direct address of this dll loaded in memory at DllBase. Now some piece of code to list all loaded dlls:

```
.386
.model flat,stdcall

  include c:\masm32\include\windows.inc
  include c:\masm32\include\kernel32.inc
  include c:\masm32\include\user32.inc

  includelib c:\masm32\lib\kernel32.lib
  includelib c:\masm32\lib\user32.lib

.const
.data
.data?
.code

  start:

    assume fs:nothing
    mov esi,fs:[030h]               ; esi = ptr on _PEB
    assume fs:error

    mov esi,[esi+0ch]               ; esi = ptr on _PEB_LDR_DATA
    lea edi,[esi+0ch]               ; edi = ptr on _PEB_LDR_DATA.InLoadOrderModuleList
    mov esi,[esi+0ch]               ; esi = flink from InLoadOrderModuleList = ptr on first
                                    ; _LDR_DATA_TABLE_ENTRY

    @list_loop:

      mov eax,dword ptr [esi+030h]          ; eax =
                                            ; LDR_DATA_TABLE_ENTRY.BaseDllName.Buffer
      invoke MessageBoxW,0,eax,eax,MB_OK    ; just show message box with dllname
      mov esi,[esi]                         ; esi = next _LDR_DATA_TABLE_ENTRY
```

```
    cmp esi,edi                          ; check if we went through all items in the list
    jne @list_loop                       ; if not take next one

  ret

end start
```

---

8. Using TEB and PEB to get kernel32.dll address

Okay now most important thing – how to get kernel32.dll address from TEB and PEB? As we have list of _LDR_DATA_TABLE_ENTRY we can go through it and just check the basedllname string with „kernel32.dll”. We can also just hash it and check hashes. But in fact (I am not for 100% sure) in xp/vista kernel32.dll is always on the third position after program.exe and ntdll.dll. As I mentioned before in windows 7 there is one extra dll („ntoskrnl.dll”) so kernel32.dll will be on 4th position. So here is piece of code:

```
.386
.model flat,stdcall

  include c:\masm32\include\windows.inc
  include c:\masm32\include\kernel32.inc
  include c:\masm32\include\user32.inc
  ;include c:\masm32\include\masm32.inc

  includelib c:\masm32\lib\kernel32.lib
  includelib c:\masm32\lib\user32.lib
  ;includelib c:\masm32\lib\masm32.lib

  ;include include\_TEB.inc

.const
.data
.data?
.code

  start:

    assume fs:nothing
    mov esi,fs:[030h]                ; esi = ptr on _PEB
    assume fs:error

    mov esi,[esi+0ch]                ; esi = ptr on _PEB_LDR_DATA
    mov esi,[esi+0ch]                ; esi = flink from InLoadOrderModuleList = ptr on first
                                     ; _LDR_DATA_TABLE_ENTRY
    mov esi,[esi]                    ; next one
    mov esi,[esi]                    ; next one
    mov esi,dword ptr [esi+018h]     ; _LDR_DATA_TABLE_ENTRY.DllBase
    ; now esi = kernel32.dll
    ret

  end start
```

I think that we can check the OSVersion in PEB so it will solve the problem with position of kernel32.dll... And there is another problem with win9x as the PEB structure on thos systems is different...

I found fragment of code which can be used to get kernel32.dll on win9x:

```
assume fs:nothing
mov eax,fs:[030h]        ; eax = PEB
mov eax, [eax + 0x34]  ; undocummented
lea eax, [eax + 0x7c]    ; undocummented
mov eax, [eax + 0x3c]  ; undocummented
```

I have no idea how it works but it should work because I found this fragment in few shellcodes. The only thing we need more is to know if we are in windows 9x or NT – we can do this as follows:

```
assume fs:nothing
mov eax,fs:[030h]        ; eax = PEB
test eax,eax
js @win_9x

@win_nt:
...
@win_9x:
…
```

Why test eax,eax? It's beacause in win 9x PEB lays in memory above address 0x80000000 and in win nt it lays above this address. If we will change 0x80000000 to binary we will get something like that: 10000....000000 – so we can see that highest bit (sign bit) is 1 – so if peb lays above this address this bit will be always 1 if below it will be always 0. And that's why we use js instruction (if sign).

By Mdew (alek.barszczewski@gmail.com)