© The Assembly-Programming-Journal, Vol. 1, No. 1 (2004)
http://www.Assembly-Journal.com

Detecting operating systems without Microsoft Advanced Programming Interface

Thomas Kruse
Universitas Virtualis

Abstract

Today nearly all programmers use the Advanced Programming Interface (API) to receive information's about given system values. By using this API's, we don't have to take care which operating system is currently available.

But sometimes it might be needful to avoid the usage of such API's. This situation is given during development of software protections in order to avoid importing functions which - eventually - will point a reverse engineer to a solution.

This essay show up a way to detect the today given operating systems from Microsoft: Windows 95, 98, ME - the non NT-based operating systems - and Windows NT4, 2000, XP, 2003 - the NT-based operating systems.

The shown source code is in Microsoft Assembler style (MASM [3]).

Keywords: *Microsoft Operating Systems, Software Protection, Assembly Programming, System Internals*

The author ***Thomas Kruse*** *has his main research focus on operating-system independent code optimising and software-protection development. He is Associate Editor of Assembly-Journal and CodeBreakers-Journal.*

Microsoft and Windows are registered trademarks of the Microsoft Corporation

I. Introduction

When analysing the structure of the Thread Environment Block (TEB - also known as Thread Information Block TIB) on different operating systems, we find additional data followed by this structure. This additional data seems to have - on non NT-based operating systems - no logical structure or length definition, where NT-based operating systems store the information inside the Process Environment Block (PEB). The only way to figure out the meaning of this data for non NT-based operating systems is to debug the same application on different operation systems[2].

II. Application start

There are many ways in detecting an operating system. It could be done by using the Windows API function *GetVersionEx[1]* and checking the version values returned in structure *OSVERSIONINFO(EX)[1]*, or by accessing Register CS. Another way is analysing the registers during start up of an application. There are several rules for them on how operating systems prepare several registers *before* executing the first instruction:

```
  Startup values for Windows 95/98/ME
  EAX == Application Entry Point
  EBX == 00530000h, a fixed value

  Startup values for Windows NT/2000/XP/2003
  EAX == NULL
  EBX == 7FFDF000h, pointer to (PEB)
```

By knowing this rules we where able to check the operating system base during start up. But then we need to store the register values of EAX and EBX for further usage - or resolve them in a different way.

A. Thread Environment Block

The TEB is prepared during application start up and contain pointers to thread related additional data. The TEB structure is available on all operating systems. It's size is defined to 34h Bytes. The TEB address could be resolved by accessing the segment register FS in the following way:

```
  assume  fs:nothing
  mov     eax,fs:[18h]
```

The register EAX will contain the base address of this block. The TEB contains - at address 18h inside the structure - a pointer to itself:

```
  pSelf          DWORD ?     ; 18h pointer to TEB/TIB
```

The last entry of TEB is the pointer to process database. On NT-based operating systems this value will point to the address of Process Environment Block (see Section II-C)

B. Additional data following TEB

Told in Section I, this additional data has no logical structure and differs on each non NT-based operating system. On Windows NT, 2000, XP and 2003 this additional Data is defined as follows:

```
NT_TEB_ADDON struct
  LastErrorValue  DWORD ?    ; 00h (34h TEB)
  LastStatusValue DWORD ?    ; 04h (38h TEB)
  CountOwnedLocks DWORD ?    ; 08h (3Ch TEB)
  HardErrorsMode  DWORD ?    ; 0Ch (40h TEB)
```

Copyright © 2004 and published by the Assembly-Programming-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
NT_TEB_ADDON ends
```

Windows 95, 98, ME didn't have such a structure; the additional data is scrambled!

C. Process Environment Block

Windows NT-based operating systems store process related data inside the Process Environment Block. The address of this structure is avaliable by accessing the segment register FS:

```
  assume  fs:nothing
  mov     eax,fs:[30h]
```

The register EAX will contain the base address of PEB.

```
  pProcess       DWORD ?     ; 30h pointer to process database
```

The version information is stored inside the PEB structure:

```
  OSMajorVersion DWORD ?     ; A4h <=4->NT / 5->2K/XP/2K3
  OSMinorVersion DWORD ?     ; A8h 0->2K / 1->XP / 2->2K3
```

D. NT-based definitions

Windows NT, 2000, XP and 2003 use fixed addresses to store PEB and TEB. The PEB is always stored at address 7FFDF000h and TEB is starting at 7FFDE000h. By knowing these two fixed values, it is possible to detect the operating system base.

III. The Trick

Section II-B has shown an add-on structure for NT-based operating systems. The data exists on non NT-based operating systems, too. But it is stored in a different way. To resolve correct memory positions - related to detect the operating system - we use a trick the analyse the additional data.

If we take a closer look to the NT_TEB_ADDON structure shown in Section II-B, we see the entry *LastErrorValue*. Nearly all Windows API's will return an error value which is accessible via *GetLastError[1]*. In addition to this, it is possible to manipulate the *LastErrorValue* via *SetLastError[1]* API. By using this API and monitoring the memory area behind the TEB, the locations for the *LastErrorValue* are:

```
  Windows 95 - TEB-base + 60h
  Windows 98 - TEB-base + 60h
  Windows ME - TEB-base + 74h
```

Now we are able to detect Windows ME or Windows 95/98. First step to our solution, but not the final one. It is possible to detect a difference between Windows 95 and Windows 98.

Section II showed the start up values and their rules. The start up value of EBX on non NT-based OS is 00530000h. Exactly this value will be found inside the additional data part - close to the now resolved *LastErrorValue*. By analysing it's location, the result will be:

```
  Windows 95 - TEB-base + 58h
  Windows 98 - TEB-base + 54h
  Windows ME - TEB-base + 7Ch
```

Now we are able to differ between each non NT-based operating system.

Copyright © 2004 and published by the Assembly-Programming-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

IV. Code creation

Right now, we where able the detect the version information for each operating system. We want an operating system independent code, we need to structure the given information's. Also it should be possible to resolve the version information *workflow independent*. Section VI will show the complete solution in Assembler.

First of all, we get the base addresses of PEB and TEB and resolve the operating system base by analysing them:

```
assume  fs:nothing
mov     ebx,fs:[18h]       ; get self pointer from TEB
mov     eax,fs:[30h]       ; get pointer to PEB / database
.if     eax==7FFDF000h && ebx==7FFDE000h
  ; WinNT based
.else
  ; Win9X based
.endif                     ; of base check NT/9X
```

The version information for NT-based operation systems is stored inside PEB. We only have to analyse the values of *OSMajorVersion* and *OSMinorVersion*:

```
  mov     ebx,[eax+0A8h]   ; get OSMinorVersion
  mov     eax,[eax+0A4h]   ; get OSMajorVersion
  .if     eax==5 && ebx==0 ; is it Windows 2000?
  .elseif eax==5 && ebx==1 ; is it Windows XP?
  .elseif eax==5 && ebx==2 ; is it Windows 2003?
  .elseif eax<=4           ; is it Windows NT?
  .endif
```

Non NT-based operating systems could be detected by analysing the additional data area behind TEB, searching the value 00530000h:

```
  mov     edx,00530000h    ; the value to search
  mov     eax,fs:[18h]     ; get the TEB base address
  mov     ebx,[eax+58h]    ; TEB-base + 58h (W95)
  mov     ecx,[eax+7Ch]    ; TEB-base + 7Ch (WME)
  mov     eax,[eax+54h]    ; TEB-base + 54h (W98)
  .if     ebx==edx         ; is it Windows 95?
  .elseif eax==edx         ; is it Windows 98?
  .elseif ecx==edx         ; is it Windows ME?
  .endif
```

V. Conclusions

Resolving the operating system by using this technique is only one possibility to avoid the usage of Advanced Programming Interface functions. Other functions, for example *GetCommandLine[1]*, *IsDebuggerPresent[1]* or named functions in this essay could be "rewritten" in the same way. In other words: the reverse engineer isn't able to set *breakpoints* on API function calls, because they didn't exist. And mixing different operating system solutions together makes life even harder for him.

References

[1] Microsoft Corporation, *Microsoft Developer Network*, http://msdm.microsoft.com
[2] Yuschuk, O., *Olly Debugger*, http://home.t-online.de/home/ollydbg
[3] Hutchenson, S., *MASM V8*, http://www.masmforum.com

Copyright © 2004 and published by the Assembly-Programming-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

VI. Appendix

```
.const
;-- return values from OS_GetOS
OS_UNKNOWN                    equ -1
OS_WIN95                      equ 1
OS_WIN98                      equ 2
OS_WINME                      equ 3
OS_WINNT                      equ 4
OS_WIN2K                      equ 5
OS_WINXP                      equ 6
OS_WIN2K3                     equ 7

.code
OS_GetOS proc
  local   _theReturnValue:DWORD
  pushad                      ; store all registers
  mov     _theReturnValue,OS_UNKNOWN
  assume  fs:nothing
  mov     ebx,fs:[18h]        ; get self pointer from TEB
  mov     eax,fs:[30h]        ; get pointer to PEB / database
  .if     eax==7FFDF000h && ebx==7FFDE000h  ; WinNT based
    mov     ebx,[eax+0A8h]    ; get OSMinorVersion
    mov     eax,[eax+0A4h]    ; get OSMajorVersion
    .if     eax==5 && ebx==0 ; is it Windows 2000?
      mov     _theReturnValue,OS_WIN2K
    .elseif eax==5 && ebx==1 ; is it Windows XP?
      mov     _theReturnValue,OS_WINXP
    .elseif eax==5 && ebx==2 ; is it Windows 2003?
      mov     _theReturnValue,OS_WIN2K3
    .elseif eax<=4           ; is it Windows NT?
      mov     _theReturnValue,OS_WINNT
    .endif
  .else                       ; Win9X based
    mov     edx,00530000h    ; the magic value to search
    mov     eax,fs:[18h]     ; get the TEB base address
    mov     ebx,[eax+58h]    ; TEB-base + 58h (W95)
    mov     ecx,[eax+7Ch]    ; TEB-base + 7Ch (WME)
    mov     eax,[eax+54h]    ; TEB-base + 54h (W98)
    .if     ebx==edx         ; is it Windows 95?
      mov     _theReturnValue,OS_WIN95
    .elseif eax==edx         ; is it Windows 98?
      mov     _theReturnValue,OS_WIN98
    .elseif ecx==edx         ; is it Windows ME?
      mov     _theReturnValue,OS_WINME
    .endif
  .endif                      ; of base check NT/9X
  popad                       ; restore all registers
  mov     eax,_theReturnValue
  ret                         ; return to caller
OS_GetOS endp
```

Copyright © 2004 and published by the Assembly-Programming-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.