

IsDebuggerPresent API

时间	作者	等级	Rank
2017-02-04 10:13:26	captain (/profile/1/)	低危	1

IsDebuggerPresent API

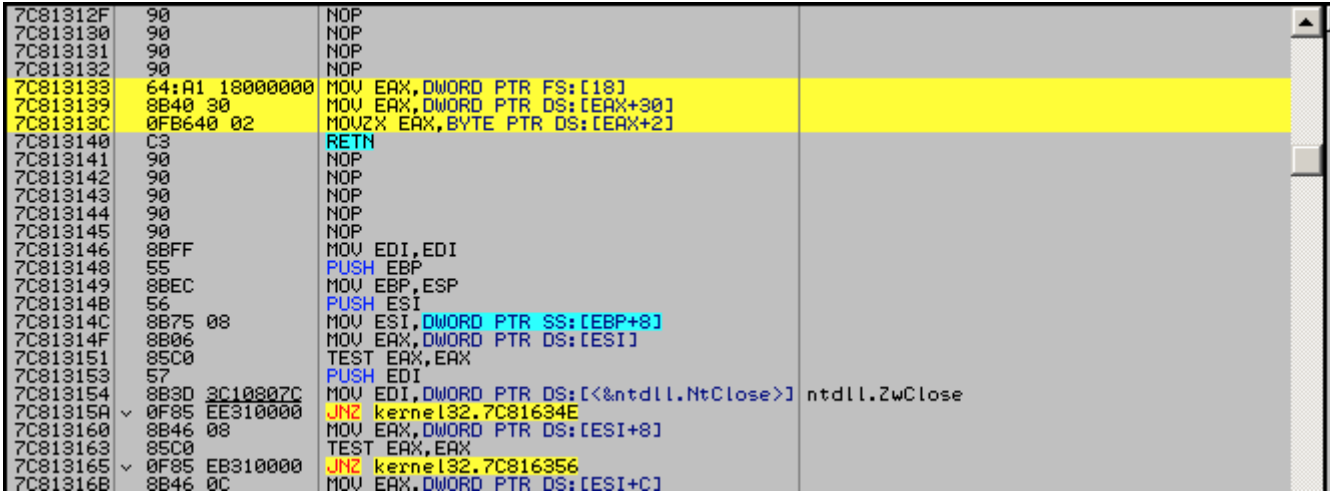
IsDebuggerPresent是一个windows api，我们可以用来检测程序是否被调试器所调试，下面是实例代码

```
/*
 * IsDebuggerPresent Example
 * Author: Osanda Malith Jayathissa (@OsandaMalith)
 * Website: https://osandamalith.wordpress.com
 */

#include <windows.h>

int main() {
    MessageBox(0, IsDebuggerPresent() ? "Debugger found" : "Debugger not found","Status",0x30);
}
```

如果我们使用调试器，将会调用message api去显示debugger found这一段话。这api到底怎么工作的？我们在调试器中调试一下这个api，你将会看到他的代码。



```
MOV EAX, DWORD PTR FS:[18]
MOV EAX, DWORD PTR DS:[EAX+30]
MOVZX EAX, BYTE PTR DS:[EAX+2]
```

FS is a special kind of segment register which contains pointers to Windows kernel data structures related to the current Process/Thread. FS:[18] is the pointer to the TIB structure. Thread Information Block is also known as the Thread Environment Block. It describes the state of the thread.

FS寄存器是一个特别的段寄存器，里面包含着当前Process/Thread中关于windows 内核数据结构的指针。FS:[18]是一个指向TIB结构的指针。线程信息块又被人们称为线程环境块。里面描述了关于当前线程的状态。

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms686708\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms686708(v=vs.85).aspx)

我们来使用windbg调试器来浏览一下TEB结构

```
ntdll!_TEB
+0x000 NtTib          : _NT_TIB
+0x01c EnvironmentPointer : (null)
+0x020 ClientId       : _CLIENT_ID
+0x028 ActiveRpcHandle : (null)
+0x02c ThreadLocalStoragePointer : 0x7ffdd02c Void
+0x030 ProcessEnvironmentBlock : 0x7ffde000 _PEB
+0x034 LastErrorValue  : 0
+0x038 CountOfOwnedCriticalSections : 0
+0x03c CsrClientThread : (null)
+0x040 Win32ThreadInfo : (null)
+0x044 User32Reserved : [26] 0
+0x0ac UserReserved   : [5] 0
+0x0c0 WOW32Reserved  : 0x77ad21dc Void
+0x0c4 CurrentLocale   : 0x409
+0x0c8 FpSoftwareStatusRegister : 0
+0x0cc SystemReserved1 : [54] (null)
+0x1a4 ExceptionCode   : 0n0
```

As you can see 0x30 is a pointer to the PEB structure which is the Process Information Block. It contains process information as the name describes. So MOV EAX,DWORD PTR DS:[EAX+30] will move the address of the PEB.

你可以看到0x30是一个指向PEB结构的指针。[https://msdn.microsoft.com/en-us/library/windows/desktop/aa813706\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/aa813706(v=vs.85).aspx)（相关资料）

里面包含了进程信息的描述。所以 MOV EAX,DWORD PTR DS:[EAX+30] 这条指令将会移动地址到PEB

```

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 SpareBytes       : [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 TlsIndex           : Uint4B
+0xe14 TlsSlots            : [256] Ptr32 Void
+0xe1c TlsIndexBits       : Uint4B
+0xe20 SpareBytes2        : [36] UChar
+0xe24 TlsIndexBits2      : Uint4B
+0xe28 TlsSlots2          : [256] Ptr32 Void
+0xe30 TlsIndexBits3      : Uint4B
+0xe34 TlsSlots3          : [256] Ptr32 Void
+0xe38 TlsIndexBits4      : Uint4B
+0xe3c TlsSlots4          : [256] Ptr32 Void
+0xe40 TlsIndexBits5      : Uint4B
+0xe44 TlsSlots5          : [256] Ptr32 Void
+0xe48 TlsIndexBits6      : Uint4B
+0xe4c TlsSlots6          : [256] Ptr32 Void
+0xe50 TlsIndexBits7      : Uint4B
+0xe54 TlsSlots7          : [256] Ptr32 Void
+0xe58 TlsIndexBits8      : Uint4B
+0xe5c TlsSlots8          : [256] Ptr32 Void
+0xe60 TlsIndexBits9      : Uint4B
+0xe64 TlsSlots9          : [256] Ptr32 Void
+0xe68 TlsIndexBits10     : Uint4B
+0xe6c TlsSlots10         : [256] Ptr32 Void
+0xe70 TlsIndexBits11     : Uint4B
+0xe74 TlsSlots11         : [256] Ptr32 Void
+0xe78 TlsIndexBits12     : Uint4B
+0xe7c TlsSlots12         : [256] Ptr32 Void
+0xe80 TlsIndexBits13     : Uint4B
+0xe84 TlsSlots13         : [256] Ptr32 Void
+0xe88 TlsIndexBits14     : Uint4B
+0xe8c TlsSlots14         : [256] Ptr32 Void
+0xe90 TlsIndexBits15     : Uint4B
+0xe94 TlsSlots15         : [256] Ptr32 Void
+0xe98 TlsIndexBits16     : Uint4B
+0xe9c TlsSlots16         : [256] Ptr32 Void
+0xea0 TlsIndexBits17     : Uint4B
+0xea4 TlsSlots17         : [256] Ptr32 Void
+0xea8 TlsIndexBits18     : Uint4B
+0xeac TlsSlots18         : [256] Ptr32 Void
+0xeb0 TlsIndexBits19     : Uint4B
+0xeb4 TlsSlots19         : [256] Ptr32 Void
+0xeb8 TlsIndexBits20     : Uint4B
+0xebc TlsSlots20         : [256] Ptr32 Void
+0xec0 TlsIndexBits21     : Uint4B
+0xec4 TlsSlots21         : [256] Ptr32 Void
+0xec8 TlsIndexBits22     : Uint4B
+0xecc TlsSlots22         : [256] Ptr32 Void
+0xed0 TlsIndexBits23     : Uint4B
+0xed4 TlsSlots23         : [256] Ptr32 Void
+0xed8 TlsIndexBits24     : Uint4B
+0xedc TlsSlots24         : [256] Ptr32 Void
+0xee0 TlsIndexBits25     : Uint4B
+0xee4 TlsSlots25         : [256] Ptr32 Void
+0xee8 TlsIndexBits26     : Uint4B
+0xeec TlsSlots26         : [256] Ptr32 Void
+0xef0 TlsIndexBits27     : Uint4B
+0xef4 TlsSlots27         : [256] Ptr32 Void
+0xef8 TlsIndexBits28     : Uint4B
+0xefc TlsSlots28         : [256] Ptr32 Void
+0xf00 TlsIndexBits29     : Uint4B
+0xf04 TlsSlots29         : [256] Ptr32 Void
+0xf08 TlsIndexBits30     : Uint4B
+0xf0c TlsSlots30         : [256] Ptr32 Void
+0xf10 TlsIndexBits31     : Uint4B
+0xf14 TlsSlots31         : [256] Ptr32 Void
+0xf18 TlsIndexBits32     : Uint4B
+0xf1c TlsSlots32         : [256] Ptr32 Void
+0xf20 TlsIndexBits33     : Uint4B
+0xf24 TlsSlots33         : [256] Ptr32 Void
+0xf28 TlsIndexBits34     : Uint4B
+0xf2c TlsSlots34         : [256] Ptr32 Void
+0xf30 TlsIndexBits35     : Uint4B
+0xf34 TlsSlots35         : [256] Ptr32 Void
+0xf38 TlsIndexBits36     : Uint4B
+0xf3c TlsSlots36         : [256] Ptr32 Void
+0xf40 TlsIndexBits37     : Uint4B
+0xf44 TlsSlots37         : [256] Ptr32 Void
+0xf48 TlsIndexBits38     : Uint4B
+0xf4c TlsSlots38         : [256] Ptr32 Void
+0xf50 TlsIndexBits39     : Uint4B
+0xf54 TlsSlots39         : [256] Ptr32 Void
+0xf58 TlsIndexBits40     : Uint4B
+0xf5c TlsSlots40         : [256] Ptr32 Void
+0xf60 TlsIndexBits41     : Uint4B
+0xf64 TlsSlots41         : [256] Ptr32 Void
+0xf68 TlsIndexBits42     : Uint4B
+0xf6c TlsSlots42         : [256] Ptr32 Void
+0xf70 TlsIndexBits43     : Uint4B
+0xf74 TlsSlots43         : [256] Ptr32 Void
+0xf78 TlsIndexBits44     : Uint4B
+0xf7c TlsSlots44         : [256] Ptr32 Void
+0xf80 TlsIndexBits45     : Uint4B
+0xf84 TlsSlots45         : [256] Ptr32 Void
+0xf88 TlsIndexBits46     : Uint4B
+0xf8c TlsSlots46         : [256] Ptr32 Void
+0xf90 TlsIndexBits47     : Uint4B
+0xf94 TlsSlots47         : [256] Ptr32 Void
+0xf98 TlsIndexBits48     : Uint4B
+0xf9c TlsSlots48         : [256] Ptr32 Void
+0xfa0 TlsIndexBits49     : Uint4B
+0xfa4 TlsSlots49         : [256] Ptr32 Void
+0xfa8 TlsIndexBits50     : Uint4B
+0xfac TlsSlots50         : [256] Ptr32 Void
+0xfb0 TlsIndexBits51     : Uint4B
+0xfb4 TlsSlots51         : [256] Ptr32 Void
+0xfb8 TlsIndexBits52     : Uint4B
+0xfbc TlsSlots52         : [256] Ptr32 Void
+0xfc0 TlsIndexBits53     : Uint4B
+0xfc4 TlsSlots53         : [256] Ptr32 Void
+0xfc8 TlsIndexBits54     : Uint4B
+0xfcc TlsSlots54         : [256] Ptr32 Void
+0xfd0 TlsIndexBits55     : Uint4B
+0xfd4 TlsSlots55         : [256] Ptr32 Void
+0xfd8 TlsIndexBits56     : Uint4B
+0xfdc TlsSlots56         : [256] Ptr32 Void
+0xfe0 TlsIndexBits57     : Uint4B
+0xfe4 TlsSlots57         : [256] Ptr32 Void
+0xfe8 TlsIndexBits58     : Uint4B
+0xfec TlsSlots58         : [256] Ptr32 Void
+0xff0 TlsIndexBits59     : Uint4B
+0xff4 TlsSlots59         : [256] Ptr32 Void
+0xff8 TlsIndexBits60     : Uint4B
+0xffc TlsSlots60         : [256] Ptr32 Void

```

This is how the PEB Structure looks like.

```

ntdll!_PEB
+0x000 InheritedAddressSpace : UChar
+0x001 ReadImageFileExecOptions : UChar
+0x002 BeingDebugged : UChar
+0x003 BitField : UChar
+0x003 ImageUsesLargePages : Pos 0, 1 Bit
+0x003 IsProtectedProcess : Pos 1, 1 Bit
+0x003 IsLegacyProcess : Pos 2, 1 Bit
+0x003 IsImageDynamicallyRelocated : Pos 3, 1 Bit
+0x003 SkipPatchingUser32Forwarders : Pos 4, 1 Bit
+0x003 IsPackagedProcess : Pos 5, 1 Bit
+0x003 IsAppContainer : Pos 6, 1 Bit
+0x003 SpareBits : Pos 7, 1 Bit
+0x004 Mutant : Ptr32 Void
+0x008 ImageBaseAddress : Ptr32 Void
+0x00c Ldr : Ptr32 _PEB_LDR_DATA
+0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
+0x014 SubSystemData : Ptr32 Void
+0x018 ProcessHeap : Ptr32 Void
+0x01c FastPebLock : Ptr32 _RTL_CRITICAL_SECTION
+0x020 AtlThunkSListPtr : Ptr32 Void
+0x024 IFEOKey : Ptr32 Void
+0x028 CrossProcessFlags : Uint4B
+0x028 ProcessInJob : Pos 0, 1 Bit
+0x028 ProcessInitializing : Pos 1, 1 Bit
+0x028 ProcessUsingVEH : Pos 2, 1 Bit
+0x028 ProcessUsingVCH : Pos 3, 1 Bit
+0x028 ProcessUsingFTH : Pos 4, 1 Bit
+0x028 ReservedBits0 : Pos 5, 27 Bits
+0x02c KernelCallbackTable : Ptr32 Void
+0x02c UserSharedInfoPtr : Ptr32 Void
+0x030 SystemReserved : [1] Uint4B
+0x034 AtlThunkSListPtr32 : Uint4B
+0x038 ApiSetMap : Ptr32 Void
+0x03c TlsExpansionCounter : Uint4B
+0x040 TlsBitmap : Ptr32 Void
+0x044 TlsBitmapBits : [2] Uint4B
+0x04c ReadOnlySharedMemoryBase : Ptr32 Void
+0x050 HotpatchInformation : Ptr32 Void
+0x054 ReadOnlyStaticServerData : Ptr32 Ptr32 Void

```

MOVZX EAX,BYTE PTR DS:[EAX+2] will move the BeingDebugged bit with zero extend to EAX. If we check the BeingDebugged bit in this context you can it's set to 1, meaning the program is being currently debugged.

MOVZX EAX,BYTE PTR DS:[EAX+2] 将把BeingDebugged位移到零扩展到EAX。如果我们在这个上下文中检查BeingDebugged位，它可以设置为1，这意味着该程序当前正在调试。

```

ntdll!_PEB
+0x000 InheritedAddressSpace : 0 ''
+0x001 ReadImageFileExecOptions : 0 ''
+0x002 BeingDebugged : 0x1 ''
+0x003 BitField : 0 ''
+0x003 ImageUsesLargePages : 0y0
+0x003 IsProtectedProcess : 0y0
+0x003 IsLegacyProcess : 0y0
+0x003 IsImageDynamicallyRelocated : 0y0
+0x003 SkipPatchingUser32Forwarders : 0y0
+0x003 IsPackagedProcess : 0y0
+0x003 IsAppContainer : 0y0
+0x003 SpareBits : 0y0
+0x004 Mutant : 0xffffffff Void
+0x008 ImageBaseAddress : 0x00400000 Void
+0x00c Ldr : 0x77c7a1e0 _PEB_LDR_DATA

```

我希望你了解IsDebuggerPresent API背后的逻辑。 这里有一个使用FASM的示例，我已经自己实现了isdebuggerpresent api。 而不是使用windows这个API，你可以使用C / C ++应用程序中的内联汇编代码来检查BeingDebugged位。

entry start

```
include 'win32a.inc'
```

```
Title      db      "Status",0
Found      db      "Debugger Found",0
NotFound   db      "Debugger Not Found",0
```

```
start:
```

```
push      0x30
push      Title
push      NotFound
push      0
call      [MessageBox]
jmp exit
```

```
push    0x10
push    Title
push    Found
push    0
call    [MessageBox]
```

```
push    0
call    [ExitProcess]
```

```
; =====
section '.idata' import data readable
```

```

; =====

library kernel32,'kernel32.dll',\
        User32,'user32.dll'

import  kernel32,\
        ExitProcess,'ExitProcess'

import  User32,\
        MessageBox,'MessageBoxA'

```

这里是一个使用C的例子，其中我首先使用ZwQueryInformationProcess内核API的PEB的地址，然后检查BeingDebugged位。API描述了使用ProcessBasicInformation作为ProcessInformationClass，我们可以检索指向PEB结构的指针。

```

NTSTATUS WINAPI ZwQueryInformationProcess(
    _In_      HANDLE      ProcessHandle,
    _In_      PROCESSINFOCLASS ProcessInformationClass,
    _Out_     PVOID       ProcessInformation,
    _In_      ULONG       ProcessInformationLength,
    _Out_opt_ PULONG      ReturnLength
);

```

检查文档你可以看到PebBaseAddress是ProcessBasicInformation结构的成员，它指向PEB。因此，我们使用ProcessBasicInformation的指针作为ProcessInformationClass参数。

```

typedef struct _PROCESS_BASIC_INFORMATION {
    PVOID Reserved1;
    PPEB PebBaseAddress;
    PVOID Reserved2[2];
    ULONG_PTR UniqueProcessId;
    PVOID Reserved3;
} PROCESS_BASIC_INFORMATION;

```

[https://msdn.microsoft.com/en-us/library/windows/desktop/ms687420\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms687420(v=vs.85).aspx)
 这里有一个例子，我使用C / C ++来获取BeingDebugged位。

```

#include <Winternl.h>
#include <Windows.h>
#include <tchar.h>

/*
 * Author: Osanda Malith Jayathissa (@OsandaMalith)
 * Website: http://OsandaMalith.wordpress.com
 * Using ZwQueryInformationProcess we get the PEB Address and
 * then we check the BeingDebugged bit to determine the process is being debugged or not.
 */

int main() {

    typedef unsigned long(__stdcall *pfnZwQueryInformationProcess)
    (
        IN HANDLE,
        IN unsigned int,
        OUT PVOID,
        IN ULONG,
        OUT PULONG
    );
    pfnZwQueryInformationProcess ZwQueryInfoProcess = NULL;

    HMODULE hNtdll = LoadLibrary(_T("ntdll.dll"));
    if (hNtdll == NULL) { }

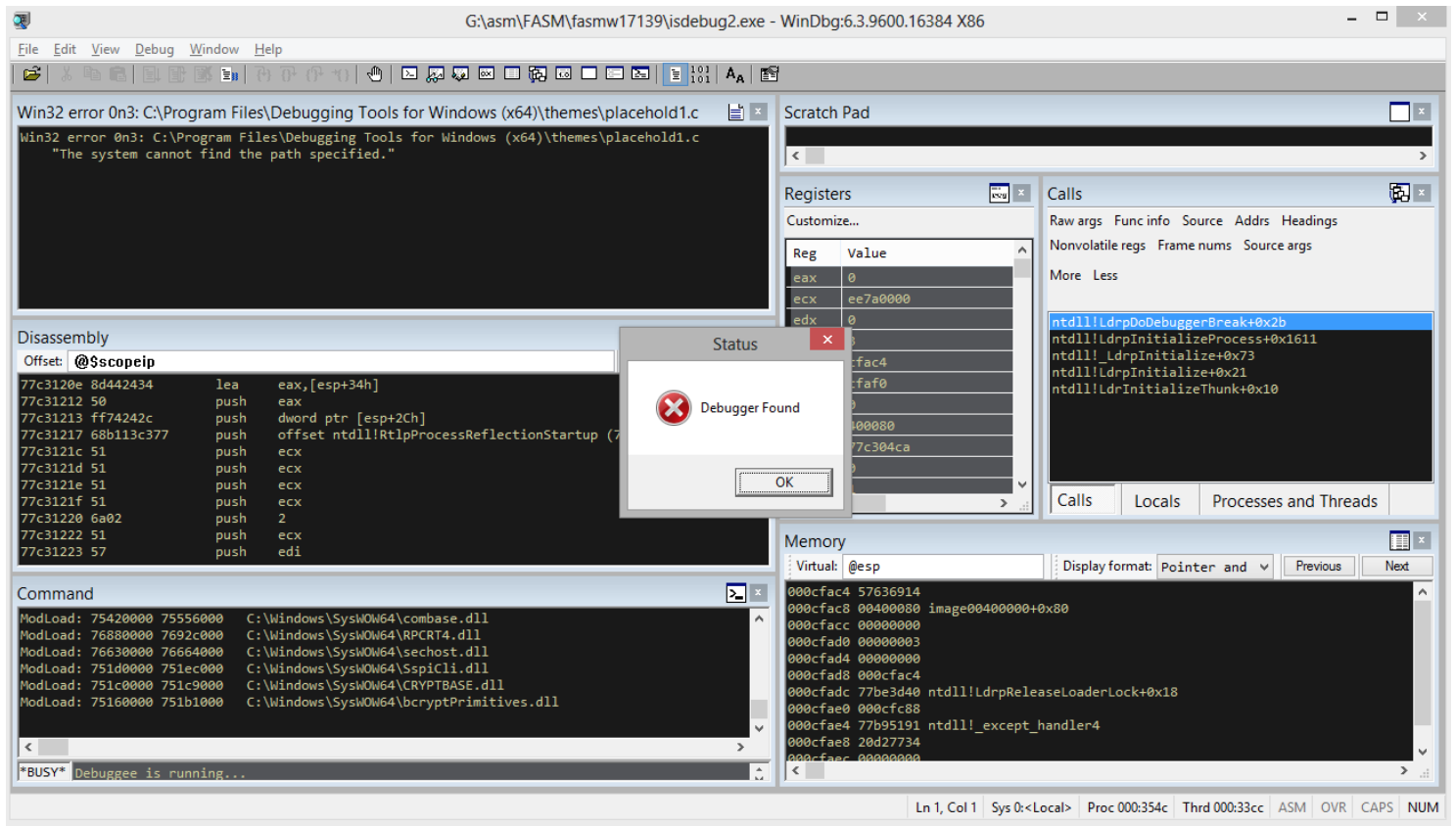
    ZwQueryInfoProcess = (pfnZwQueryInformationProcess) GetProcAddress(hNtdll,
        "ZwQueryInformationProcess");
    if (ZwQueryInfoProcess == NULL) { }
    unsigned long status;

    DWORD pid = GetCurrentProcessId();
    HANDLE hProcess = OpenProcess(PROCESS_QUERY_INFORMATION, FALSE, pid);
    PROCESS_BASIC_INFORMATION pbi;
    status = ZwQueryInfoProcess(hProcess,
                                ProcessBasicInformation,
                                &pbi,
                                sizeof(pbi),
                                NULL);

    PPEB peb_addr = pbi.PebBaseAddress;
    DWORD ptr = pbi.PebBaseAddress;
    ptr|=0x2;
    DWORD *temp = ptr;
    MessageBox(0, *temp & 1 ? "Debugger found" : "Debugger not found","Status",0x30);

    return 0;
}

```

审核评价： 没有任何评价...

评论

提交