

Defeating IsDebuggerPresent()

Notebook: Reverse Engineering

Created: 1/09/2020 6:08 PM

Author: Daniel de Jager

Updated: 1/09/2020 6:31 PM

Defeating IsDebuggerPresent()

A popular anti-forensic technique used by malware authors is to detect whether their applications are being debugged. It is accomplished by using kernelbase.dll (kernel32) function called IsDebuggerPresent() and IsRemoteDebuggerPresent(). This document will focus on defeating IsDebuggerPresent().

The methodology used is as follows:

- Write a C++ program that checks whether a debugger is present
- Reverse the PE file and change program flow by patching

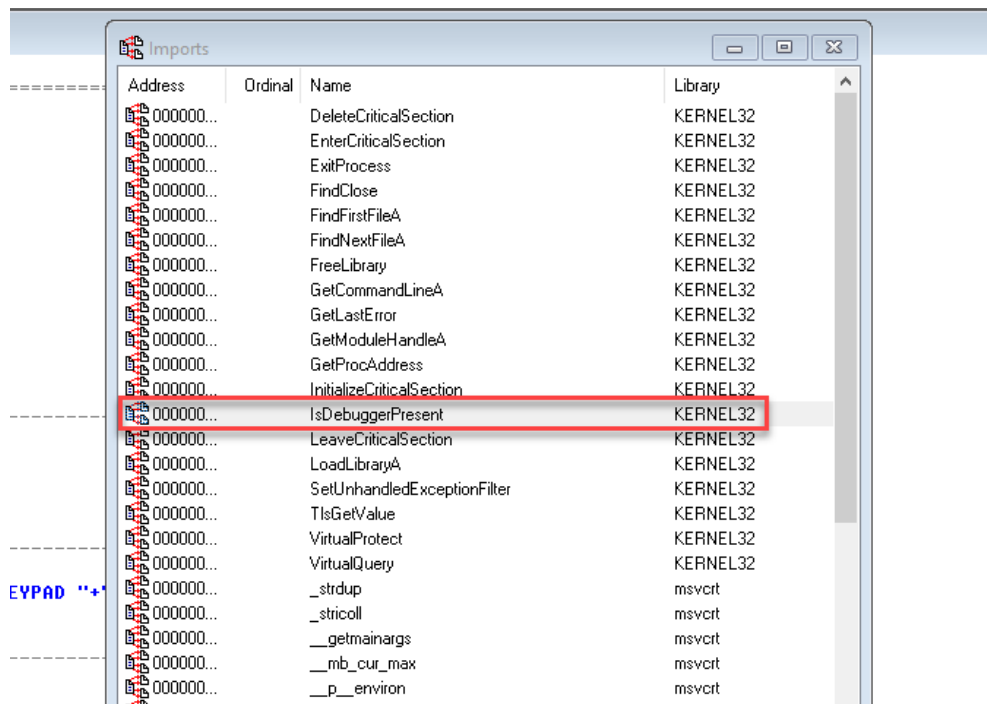
IsDebuggerPresent() Code:

```
#include <iostream>
#include <windows.h>

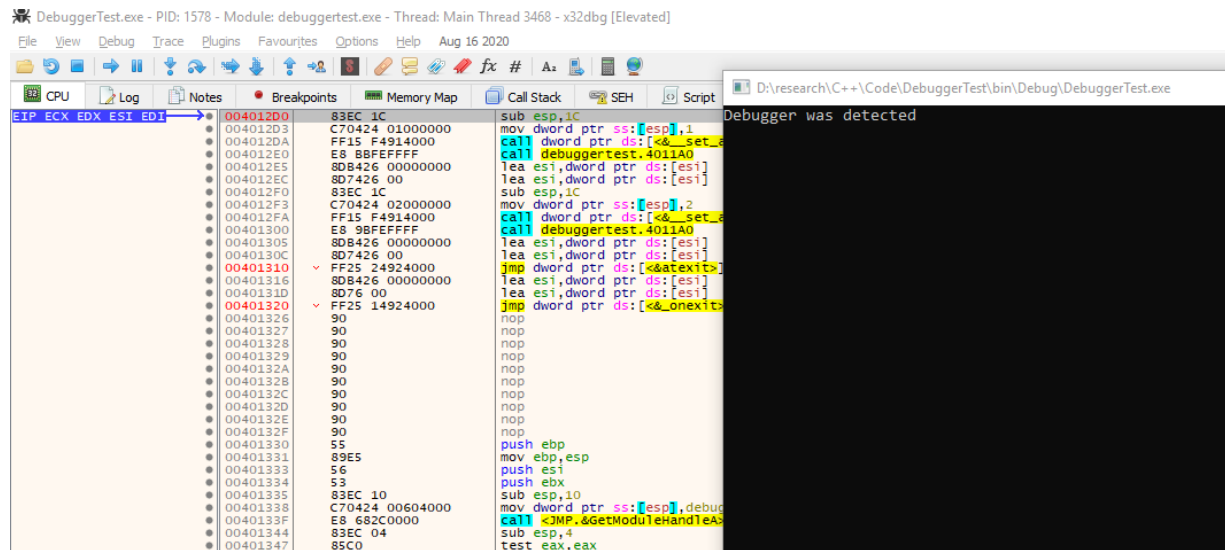
using namespace std;

int main()
{
    if(IsDebuggerPresent())
    {
        std::cout << "Debugger was detected" << endl;
    }
    else
    {
        std::cout << "Nothing to worry about" << endl;
    }
    std::cin.get();
    return 0;
}
```

- Performing input analysis with IDA shows that the function is imported



- Testing the application in x32DBG shows that the application does indeed detect the debugger



Under the hood of kernelbase.dll

- IsDebuggerPresent() is only three lines of code
- It reads as follows:
 - take the value of the FS register at address 0x30 and store it in EAX
 - take the byte value of eax offset with 2 bytes and replace EAX
 - return

```
.text:1010AAD0
.text:1010AAD0 ; BOOL __stdcall IsDebuggerPresent()
.text:1010AAD0 public IsDebuggerPresent
.text:1010AAD0 IsDebuggerPresent proc near ; CODE XREF: sub_101408E5:loc_10140918↓p
* .text:1010AAD0 mov eax, large fs:30h
* .text:1010AAD6 movzx eax, byte ptr [eax+2]
* .text:1010AADA retn
.text:1010AADA IsDebuggerPresent endp
.text:1010AADA
.text:1010AADA ; -----
```

https://en.wikipedia.org/wiki/Win32_Thread_Information_Block#:~:text=In%20computing%2C%20the%20Win32%20Thread,similar%20structure%20in%20OS%2F2.

- FS is a datastructure in Win32 x86 that stores information about the currently running thread - also known as the Thread Environment Block (TEB)
- The TIB can be used get loads of information about the process without calling Win32 API
- TIBx30 points to the process environment block

Bytes/ Type	offset (32- bit, FS)	offset (64- bit, GS)	Windows Versions	Description
pointer	FS:[0x00]	GS:[0x00]	Win9x and NT	Current Structured Exception Handling (SEH) frame Note: the 64-bit version of Windows uses stack unwinding done in kernel mode instead.
pointer	FS:[0x04]	GS:[0x08]	Win9x and NT	Stack Base / Bottom of stack (high address)
pointer	FS:[0x08]	GS:[0x10]	Win9x and NT	Stack Limit / Ceiling of stack (low address)
pointer	FS:[0x0C]	GS:[0x18]	NT	SubSystemTib
pointer	FS:[0x10]	GS:[0x20]	NT	Fiber data
pointer	FS:[0x14]	GS:[0x28]	Win9x and NT	Arbitrary data slot
pointer	FS:[0x18]	GS:[0x30]	Win9x and NT	Linear address of TEB
End of NT subsystem independent part; below are Win32-dependent				
pointer	FS:[0x1C]	GS:[0x38]	NT	Environment Pointer
pointer	FS:[0x20]	GS:[0x40]	NT	Process ID (in some Windows distributions this field is used as 'DebugContext')
4	FS:[0x24]	GS:[0x48]	NT	Current thread ID
4	FS:[0x28]	GS:[0x50]	NT	Active RPC Handle
4	FS:[0x2C]	GS:[0x58]	Win9x and NT	Linear address of the thread-local storage array
4	FS:[0x30]	GS:[0x60]	NT	Linear address of Process Environment Block (PEB)
4	FS:[0x34]	GS:[0x68]	NT	Last error number
4	FS:[0x38]	GS:[0x6C]	NT	Count of owned critical sections
4	FS:[0x3C]	GS:[0x70]	NT	Address of CSR Client Thread
4	FS:[0x40]	GS:[0x78]	NT	Win32 Thread Information
124	FS:[0x44]	GS:[0x80]	NT, Wine	Win32 client information (NT), user32 private data (Wine), 0x60 = SetLastError (Win95&98), 0x74 = SetLastError (WinME)
4	FS:[0xC0]	GS:[0x100]	NT	Reserved for Wow64. Contains a pointer to FastSysCall in Wow64.
4	FS:[0xC4]	GS:[0x108]	NT	Current Locale
4	FS:[0xC8]	GS:[0x10C]	NT	FP Software Status Register

<https://docs.microsoft.com/en-us/windows/win32/api/winternl/ns-winternl-peb>

PEB+2 points to BeingDebugged()

Syntax

```

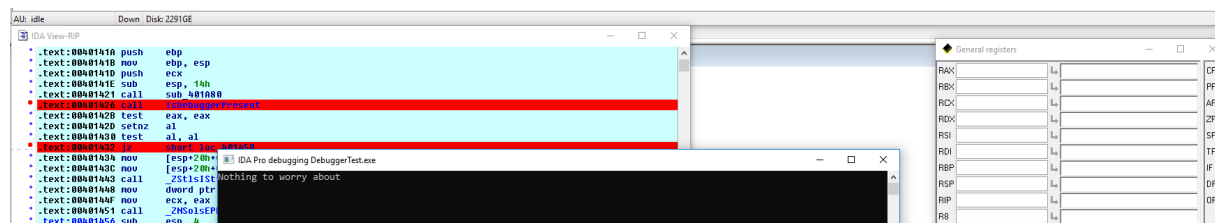
C++
Copy

typedef struct _PEB {
    BYTE Reserved1[2];
    BYTE BeingDebugged;
    BYTE Reserved2[1];
    PVOID Reserved3[2];
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    PVOID Reserved4[3];
    PVOID AtlThunkSListPtr;
    PVOID Reserved5;
    ULONG Reserved6;
    PVOID Reserved7;
    ULONG Reserved8;
    ULONG AtlThunkSListPtr32;
    PVOID Reserved9[45];
    BYTE Reserved10[96];
    PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
    BYTE Reserved11[128];
    PVOID Reserved12[1];
    ULONG SessionId;
} PEB, *PPEB;

```

Debugging in IDA

- If a change the value of RAX to 0, then I defeat the logic and a debugger is not said not to be present



With x32DBG

Here I change the process flow (changed opco JE to JNE) using assembly

Procedure:

- Alt+E
- Click on Application and search for IsDebuggerPresent
- Double Click
- Follow in Assembly
- Press F2 to create a breakpoint
- Execute until the breakpoint is reached then make the changes

Address	Disassembly	Comment
0040141E	82E5 14	sub esp,14
00401421	E8 5A060000	call debuggertest.401A80
00401426	E8 69280000	call <JMP.&IsDebuggerPresent>
0040142B	85C0	test eax,eax
0040142D	0F95C0	setne al
00401430	84C0	test al,al
00401432	75 27	jne debuggertest.40145B
00401434	C74424 04 45604000	mov dword ptr ss:[esp+4],debuggertest.4
0040143C	C70424 804AF46F	mov dword ptr ss:[esp],<libstdc++-6._ZS
00401443	E8 AC000000	call <JMP.&ZSt15Istl1char_traitsICEERS
00401448	C70424 FC144000	mov dword ptr ss:[esp],<JMP.&ZSt4endl1
0040144F	89C1	mov ecx,eax
00401451	E8 BE000000	call <JMP.&ZNSo15EPFRSoS_E>
00401456	83EC 04	sub esp,4
00401459	E8 25	jmp debuggertest.401480
0040145B	C74424 04 58604000	mov dword ptr ss:[esp+4],debuggertest.4
00401463	C70424 804AF46F	mov dword ptr ss:[esp],<libstdc++-6._ZS
0040146A	E8 85000000	call <JMP.&ZSt15Istl1char_traitsICEERS
0040146F	C70424 FC144000	mov dword ptr ss:[esp],<JMP.&ZSt4endl1
00401476	89C1	mov ecx,eax
00401478	E8 97000000	call <JMP.&ZNSo15EPFRSoS_E>
0040147D	83EC 04	sub esp,4
00401480	B9 A048F46F	mov ecx,<libstdc++-6._ZSt3cin>

Comments on the right side of the assembly window:

- [esp+4]: "Nothing to worry about", 406045: "Debugger was detected"
- ecx: "&Ec00%cao"
- [esp+4]: "Nothing to worry about", 40605B: "Nothing to worry about"
- ecx: "&Ec00%cao"
- ecx: "&Ec00%cao", 6FF448A0: "ø-øo"

Outcome:

Address	Disassembly	Comment
0040141B	89E5	mov ebp,esp
0040141D	51	push ecx
0040141E	83EC 14	sub esp,14
00401421	E8 5A060000	call debuggertest.401A80
00401426	E8 69280000	call <JMP.&IsDebuggerPresent>
0040142B	85C0	test eax,eax
0040142D	0F95C0	setne al
00401430	84C0	test al,al
00401432	75 27	jne debuggertest.40145B
00401434	C74424 04 45604000	mov dword ptr ss:[esp+4],deb
0040143C	C70424 804AF46F	mov dword ptr ss:[esp],<libs
00401443	E8 AC000000	call <JMP.&ZSt15Istl1char_t
00401448	C70424 FC144000	mov dword ptr ss:[esp],<JMP.
0040144F	89C1	mov ecx,eax
00401451	E8 BE000000	call <JMP.&ZNSo15EPFRSoS_E>
00401456	83EC 04	sub esp,4
00401459	E8 25	jmp debuggertest.401480
0040145B	C74424 04 58604000	mov dword ptr ss:[esp+4],deb
00401463	C70424 804AF46F	mov dword ptr ss:[esp],<libs
0040146A	E8 85000000	call <JMP.&ZSt15Istl1char_t
0040146F	C70424 FC144000	mov dword ptr ss:[esp],<JMP.
00401476	89C1	mov ecx,eax

Output window (D:\research\C++\Code\DebuggerTest\bin\Debug\DebuggerTest.exe):

```
Nothing to worry about
```