An Analysis of IsDebugerPresent

Author: Thomas Thelen

Date: 13 October 2015

Motivation

One way of keeping software safe from crackers and intellectual software thieves involves checking for the presence of a debugger. Although good in intention, a malware author may try to thwart the efforts of the reverse engineer by employing such a technique. Being able to bypass such checks are essential for further dissection and dynamic analysis of the malicious code.

Microsoft provides a function in kernel32.dll named IsDebuggerPresent. It is also accessible through the common windows header file [4]. The function follows,

```
BOOL WINAPI IsDebuggerPresent(void);
```

It will return 0 if a debugger is detected and may be any other number otherwise. If a debugger is detected, the programmer way direct flow accordingly. In order to understand how to defeat this protection, a closer inspection is taken with x64dbg.

Theory

The thread information block (TEB), shown in Figure 1, is a structure that holds information pertaining to a processes' current thread [5]. It contains information such as the SEH frame base and pointers to additional data structures. One such structure is the Process Environment Block (PEB). The PEB contains structures related to the running process [1]. One of its elements, at offset 0x02 is the "BeingDebugged" flag. This flag is set to 1 when a debugger is present and 0 otherwise.

Thread Environmental Block	
0x00	NtTib
0x1c	*EnvironmentalPointer
0x20	ClientId
0x28	*ActiveRpcHandle
0x2c	*ThreadLocalStoragePointer
0x30	*ProcessEnvironmentalBlock
0x34	LastErrorValue
0x38	CountOfOwnedCriticalSections
0x3c	*CsrClientThread
0x40	*Win32ThreadInfo
0x44	User32Reserved[26]
0xac	UserReserved[5]
0xc0	*WOW32Reserved
0xc4	CurrentLocale
0xc8	FpSoftwareStatusRegister
0xcc	*SystemReserved1[54]
0x1a	ExceptionCode
0x1a8	ActivationContextStack
0x1bc	SpareBytes1[2]
0x1d4	GdiTebBatch

Figure1.. The Thread Environment Block has 21 different fields.

Process Environment Block	
0x000	IneritedAddressSpace
0x001	REadImageFileExecOptions
0x002	BeingDebugged

Figure2..A small portion of the PEB. The PEB has 66 different fields.

It will be shown that the IsDebuggerPresent function acts as a wrapper for a subroutine that first accesses the Thread Environment Block to find the location of the Process Environment Block. Once the address has been found, it reads the byte located in BeingDebugged.

Disassembly

A step inside kernel32.IsDebuggerPresent shows a disassembly with a total of four instructions.

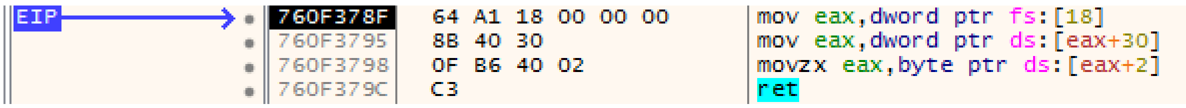

Figure 3. There are four instructions inside the kernel32 module.

It is clear that the first instruction is going to move the contents at 0x18 of the TEB into eax. A pointer to the base of the TEB resides at that particular offset [1]. This will serve as a base address when locating the address of the PEB.

With the TEB base loaded into eax, the second instruction loads the value at offset 0x30. Referring back to Figure 1, the value at 0x30 is an address to the base of the PEB. The instruction moves the address into eax and continues execution.

The third instruction loads a single byte from the second offset in the PEB. According to Figure 2, this is the BeingDebugged flag. The value is moved into the eax register before returning to the program's calling module.

Usage

It is common to see the execution of the program to change after the call. In all cases the eax register is checked and then a jump is taken to the appropriate code section.

```
EIP ──────────→ • 0040153D   83 F8 01    cmp eax,1
                • 00401540   0F 94 C0    sete al
                • 00401543   84 C0       test al,al
```

Figure 4. The testing algorithm.

The first instruction compares eax with 1. This will set the zero flag in the case that a debugger is attached. The zero flag can be checked using a multitude of opcodes however in this case, a test is used at the end.

Once the comparison is made and zero flag's status determined, the value of Al is modified. According to the Intel Instruction Set, sete will set the specified register to 0 if the zero flag is not set, and 1 if it is. [3] In this context, it sets the value of the zero flag to al.

The last instruction performs a test on the newly set al register. The test opcode sets the zero flag if the test was successful otherwise leaves it unchanged. This test will only set the zero flag if the al register is set to one, which can only happen if a debugger is attached.

From here it is up to the programmer on how to direct program flow. The program might have a cleanup routine and then exit, behave differently, or wait for the debugger to close. In all cases a jump will be taken based off of the previous test instruction.

```
EIP ───→ 00401545  74 2B                    je isdebugpresent.401572
         00401547  C7 44 24 0C 00 00 00 00  mov dword ptr ss:[esp+C],0
         0040154F  C7 44 24 08 00 F0 47 00  mov dword ptr ss:[esp+8],isdebugpresent  47F000:"Error: Debugger"
         00401557  C7 44 24 04 10 F0 47 00  mov dword ptr ss:[esp+4],isdebugpresent  47F010:"Debugger Detected"
         0040155F  C7 04 24 00 00 00 00     mov dword ptr ss:[esp],0
         00401566  A1 E8 94 48 00           mov eax,dword ptr ds:[<&MessageBoxA>]
         0040156B  FF D0                    call eax
         0040156D  83 EC 10                 sub esp,10
         00401570  EB 29                    jmp isdebugpresent.40159B
         00401572  C7 44 24 0C 00 00 00 00  mov dword ptr ss:[esp+C],0
         0040157A  C7 44 24 08 22 F0 47 00  mov dword ptr ss:[esp+8],isdebugpresent  47F022:"Okay!"
         00401582  C7 44 24 04 28 F0 47 00  mov dword ptr ss:[esp+4],isdebugpresent  47F028:"No Debugger Found"
         0040158A  C7 04 24 00 00 00 00     mov dword ptr ss:[esp],0
         00401591  A1 E8 94 48 00           mov eax,dword ptr ds:[<&MessageBoxA>]
         00401596  FF D0                    call eax
         00401598  83 EC 10                 sub esp,10
         0040159B  B8 00 00 00 00           mov eax,0
         004015A0  8B 4D FC                 mov ecx,dword ptr ss:[ebp-4]
         004015A3  C9                       leave
         004015A4  8D 61 FC                 lea esp,dword ptr ds:[ecx-4]
         004015A7  C3                       ret
```

Figure 5. Controlling program flow after the test.

Figure 5 shows code that takes a jump to a locations that display if a debugger is detected.

Bypassing

There are numerous ways to bypass the function to continue reversing the binary.

Temporary Changes

1. Inside the kernel32 module change an instruction to move 0 into the eax register.

```
EIP → 760F378F   64 A1 18 00 00 00   mov eax,dword ptr fs:[18]
      760F3795   8B 40 30            mov eax,dword ptr ds:[eax+30]
      760F3798   B8 00 00 00 00      mov eax,0
      760F379D   C3                  ret
```

Figure 6. Changing the method from kernel32.

   With 0 in eax, the cmp instruction will not set the zero flag, resulting in a jump away from the exit code. This will change the size and potentially overwrite the ret opcode.

2. Set the Zero Flag after the test instruction. This will emulate a test resulting from a debugger being absent. The jump away from the exit code is then taken.

Permanent Changes

1. Manipulate the jump conditional statement. This will always jump to the desired section of code, regardless of a debugger's presence.

```
EIP → 00401545   EB 2B                     jmp isdebugpresent.401572
      00401547   C7 44 24 0C 00 00 00 00   mov dword ptr ss:[esp+C],0
      0040154F   C7 44 24 08 00 F0 47 00   mov dword ptr ss:[esp+8],isdebugpresent   47F000:"Error: Debugger"
      00401557   C7 44 24 04 10 F0 47 00   mov dword ptr ss:[esp+4],isdebugpresent   47F010:"Debugger Detected"
      0040155F   C7 04 24 00 00 00 00      mov dword ptr ss:[esp],0
      00401566   A1 E8 94 48 00            mov eax,dword ptr ds:[<&MessageBoxA>]
      0040156B   FF D0                     call eax
      0040156D   83 EC 10                  sub esp,10
      00401570   EB 29                     jmp isdebugpresent.40159B
      00401572   C7 44 24 0C 00 00 00 00   mov dword ptr ss:[esp+C],0
      0040157A   C7 44 24 08 22 F0 47 00   mov dword ptr ss:[esp+8],isdebugpresent   47F022:"Okay!"
      00401582   C7 44 24 04 28 F0 47 00   mov dword ptr ss:[esp+4],isdebugpresent   47F028:"No Debugger Found"
      0040158A   C7 04 24 00 00 00 00      mov dword ptr ss:[esp],0
      00401591   A1 E8 94 48 00            mov eax,dword ptr ds:[<&MessageBoxA>]
      00401596   FF D0                     call eax
```

Figure 7. The je opcode has been changed to jmp.

2. Change the compare statement to check eax against zero. This will not set the zero flag, reversing the outcome. The test instruction will set the zero flag resulting in the jump being taken.

```
0040153D   83 F8 00   cmp eax,0
00401540   0F 94 C0   sete al
00401543   84 C0      test al,al
```

Figure 8. Instead of comparing eax against 1, it can be compared to 0.

3. Move 0 into the al register. Doing so will result in the test being true and the setting of the zero flag.

```
EIP → 0040153D   83 F8 01   cmp eax,1
      00401540   B0 00      mov al,0
      00401542   90         nop
      00401543   84 C0      test al,al
```

Figure 9. The sete opcode has been overwritten with a mov and nop.

Conclusions

The process of locating the BeingDebugged byte is shown below in Figure 10. First, the eax register holds the address of the TEB. Then, the address of the PEB is placed into it. Finally, the value of BeingDebugged, offset 0x002 of the PEB, is moved into eax.

TEB	
0x..	
0x030	*ProcessEnvironmentalBlock
0x..	

PEB	
0x...	
0x002	BeingDebugged
0x..	

Figure 10. Data structures used.

The routine then returns control back to the calling module.

One nuance to take note of is that patching cannot occur in kernel32. Once the call is made, the program shifts to the IsDebuggerPresent module, shown in Figure 11.

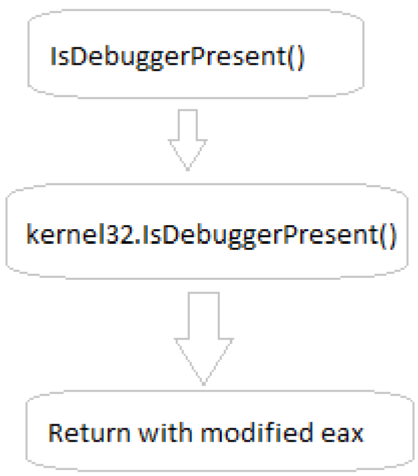

Figure11. IsDebuggerPresent Flow

Once the program exits with the new eax value, the control is passed back to the calling module. In most cases this is the executable of interest.

Bypassing the call is an easy task that offers a multitude of ways. These range from permanent program modification to temporary flag changing. Plugins also exist for most debuggers that offer extensive hiding features.

Woks Cited

[1] "Win32 Thread Environment Block." Shit We Found out. 10 Oct. 2013. Web. 13 Oct. 2015. <http://shitwefoundout.com/wiki/Win32_Thread_Environment_Block>.

[2] "PEB Structure." Microsoft Windows. Microsoft. Web. 13 Oct. 2015. <https://msdn.microsoft.com/en-us/library/windows/desktop/aa813706(v=vs.85).aspx>.

[3] "Intel Instruction Set - SETE/SETZ." Intel Instruction Set - SETE/SETZ. Web. 13 Oct. 2015. <http://web.itu.edu.tr/kesgin/mul06/intel/instr/sete_setz.html>.

[4] "IsDebuggerPresent Function." Microsoft Windows. Microsoft. Web. 13 Oct. 2015. <https://msdn.microsoft.com/en-us/library/windows/desktop/ms680345(v=vs.85).aspx>.

[5] "PEB-Process-Environment-Block." Aldeid. 27 May 2015. Web. 13 Oct. 2015. <http://www.aldeid.com/wiki/PEB-Process-Environment-Block>.