

1st Ed

Windows Anti-Debugging & Anti-Anti-Debugging Techniques

Windows Anti-Debugging & Anti-Anti-Debugging Techniques

The world is a dangerous place.

Not because of those who do evil,

but because of those that stand by and do nothing.

Contents

Why?4

About the Author.....4

What about Linux?.....4

Can I use the content in other ways?4

Technique 1 – IsDebuggerPresent().....5

 How to Bypass6

Why?

As I got obsessed with documenting everything I learn in a way that if I lose my memory, I can learn them again from the ground up, I decided to prepare a write up on Anti-Debugging and Anti-Anti-Debugging Techniques performed on Windows operating system. This paper will not include pseudo code like lots of other blogs or paper you see (or at least I see.) but it will provide proof of concept code in C++ with hands-on examples. This paper will eventually turn into a book as I add new content because these techniques will never end and depending on your knowledge base and creativity, you can invent your own.

About the Author

It's me again, Arash TC, (the low level security dude :D). One thing you definitely agree on about me (hope so) is that I don't spam the reader with unnecessary bullshit telling people about the history of assembly language or how my high school friend wrote a kickass virus with library's computer. You open the book, you jump into the actual content. Free of charge as usual because who am I to charge people for gaining knowledge. Remember to Learn and Contribute.

What about Linux?

The Linux part will be handled by my friend Kami¹ and the content will be uploaded (or maybe is uploaded by the time you're reading this) in the same github repository.²

Can I use the content in other ways?

Yes. Whatever creative commons lets you to. Specific license file is in the repository.³

¹ <https://twitter.com/k4m1>

² <https://github.com/Captainarash/Reverse-Engineer/tree/master/Anti-Debugging%20and%20Anti-Anti-Debugging/Linux>

³ <https://github.com/Captainarash/Reverse-Engineer/blob/master/LICENSE>

Technique 1 – IsDebuggerPresent()

IsDebuggerPresent() is a function available in kernel32.dll, which checks a variable in the PEB structure of the current process. Now you may ask what is PEB? PEB is short for Process Environment Block which contains very useful information about a process. Each process has its own PEB structure loaded in memory. Here is a picture on the left I found on the internet to give you an idea of PEB. On the right, there is the specific variables (flags) defined in PEB found on MSDN⁴.

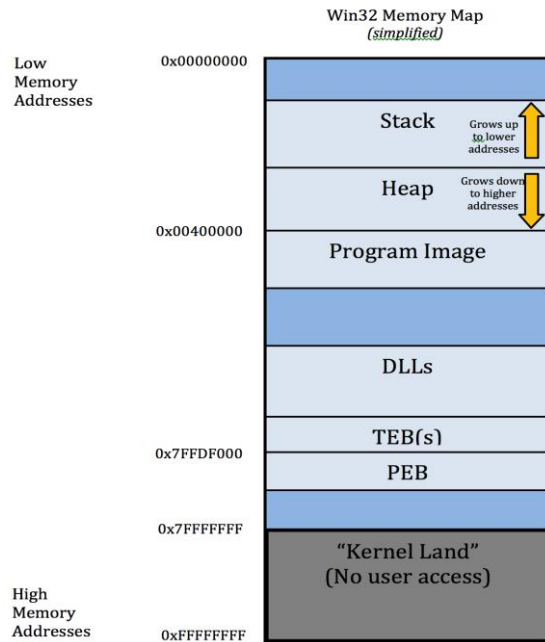


Figure 1- 1

```
typedef struct _PEB {
    BYTE Reserved1[2];
    BYTE BeingDebugged;
    BYTE Reserved2[1];
    PVOID Reserved3[2];
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    BYTE Reserved4[104];
    PVOID Reserved5[52];
    PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
    BYTE Reserved6[128];
    PVOID Reserved7[1];
    ULONG SessionId;
} PEB, *PPEB;
```

Figure 1- 2

As you can see, at offset 2 into PEB structure there is BeingDebugged flag. If the process is created inside a debugger, this flag will be set to 1. IsDebuggerPresent() function checks this flag and returns 1 if it's inside a debugger. Here's the code sample you can test inside and outside a debugger and see the results:

```
#include <cstdlib>

#include <Windows.h>

#include <stdio.h>

#include <iostream>

using namespace std;
```

⁴ You can read more about PEB on MSDN [https://msdn.microsoft.com/library/aa813706\(VS.85\).aspx](https://msdn.microsoft.com/library/aa813706(VS.85).aspx)

```
int main(int argc, char** argv) {  
    if (IsDebuggerPresent())  
    {  
        printf("Go home kiddo!");  
        return -1;  
    }  
    else {  
        printf("Good Boy!");  
        std::cin.ignore();  
        return 0;  
    }  
}
```

Compiling and running the above code inside a debugger results in printing “Go Home Kiddo!” and exiting with error.

How to Bypass

Let’s see what happens inside a debugger. We open the executable inside Olly and then search for all referenced text strings⁵. You can also find the `IsDebuggerPresent()` more efficiently by looking for its function call in all intermodular calls⁶. Anyways, after finding the address where the call to `IsDebuggerPresent()` happens, we set a breakpoint and let the program run. Look at figure 1-3 on the next page. Stepping into the call, we reach a point in `kernel32.dll` where it will actually check for `BeingDebugged` flag in PEB structure (Figure 1-4). PEB structure can be found at offset 30 into FS segment register. After that, we can find `BeingDebugged` flag at offset 2 into the PEB structure.

⁵ Right-click on the disassembler -> Search for -> All referenced text strings

⁶ Right-click on the disassembler -> Search for -> All intermodular calls

[illegible]



Figure 1-5

Beginning of the execution we can go to the address fs:[30] (figure 1-6). Then we follow that address in dump and we can see at the offset 2 of that address, the value 1 is set. We are currently looking at PEB structure in memory dump view of Olly (figure 1-7) and we can set it to 0 so IsDebuggerPresent() will always return false (figure 1-8). This flag is an informative flag and does not affect the execution of the program.

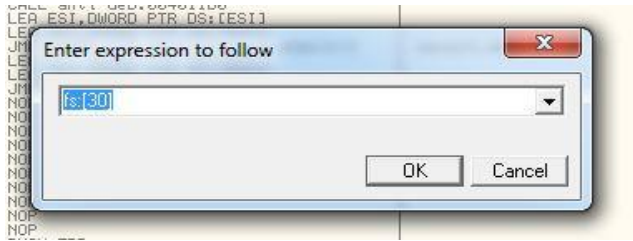


Figure 1-6

Address	Hex dump	ASCII
7EFDE000	00 00 01 00 FF FF FF FF 00 00 40 00 00 02 C6 77	..0.fw
7EFDE010	30 14 71 00 00 00 00 00 00 00 71 00 00 21 C6 77	0q.....q..fw
7EFDE020	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 000.....
7EFDE030	00 00 00 00 00 00 00 00 00 00 04 00 00 00 00 004.....
7EFDE040	50 42 C6 77 03 00 00 00 00 00 00 00 00 00 00 00	PBfw.....
7EFDE050	00 00 00 00 90 0A FE 7E 00 00 FB 7E 28 02 FC 7E	...E..".J"(0h

Figure 1-7

Address	Hex dump	ASCII
7EFDE000	00 00 00 00 FF FF FF FF 00 00 40 00 00 02 C6 77fw
7EFDE010	30 14 71 00 00 00 00 00 00 00 71 00 00 21 C6 77	0q.....q..fw
7EFDE020	00 00 00 00 00 00 00 00 01 00 00 00 00 00 00 000.....

Figure 1-8