

Anti Debugging Techniques

By Sriram.P,
Available at www.triyag.com

SNo	Topic	Page No
1	Introduction	2
2	Specific Debugger Detection Techniques	2
2.0	Olly Debugger Detection Techniques	2
2.1	Crash the Olly with OutputDebugString()	2
2.2	Olly Prefix Method	3
2.3	Olly Memory PAGE_GUARD handler	3
2.4	Olly ESI technique	3
3.0	SoftICE detection Techniques	3
3.1	Detection Thro CreateFileA()	3
3.2	Detection Thro EnumDeviceDrivers()	4
4.0	Generic Debugger Detection Techniques	4
4.1	IsDebuggerPresent() API	4
4.2	PEB Debugger Check	4
4.3	csrss.exe module debug method	4
4.4	OutputDebugString() Method	5
4.5	BlockInput()	5
4.6	GetTickCount() Time based attack	5
4.7	CloseHandle() and NTClose()	5
4.8	INT3 or 0xCC instruction	5
4.9	Interrupt 2Dh	6
4.10	Setting the Trap Flag bit	6
4.11	PUSHSS, POPSS and PUSHF	6
4.12	RDTSC – Time Keeping	7
4.13	File And Memory CRC	7
5.0	Anti-Virtualization Techniques	7
5.1	Detecting presence of VMWare	7
5.1.1	Red Pill Technique	7
5.1.2	SLDT Method	8
5.1.3	Using Privilege Instruction OUT and IN	8
5.2	Skipping Nod32 Emulation	8
5.3	Detecting Wine And Honey pots	8
5.4	Detecting sandboxes	8
6.0	Conclusions	9
7.0	Bibliography	9

Abstract

Anti-Debugging techniques take different forms from **hiding code from reverse engineers** and also to **avoiding programs from automated analysis in virtual environments**. Gone were days where the malwares encryption used XOR or some algorithms implemented like LZMA in UPX or used simple API to check debugger detection. My day-to-day experience with malwares through Comodo Antivirus for 3 years not only enabled me to keep myself updated with the latest anti-debugging techniques by different malwares and protectors but also made me write Anti-Anti debugging techniques. In this paper I would like to share my research on the currently prevailing anti-debugging techniques used by various commercial software like ASProtect, Armadillo, Themida, SVKP, VMProtect, and some of the famous malwares tricks based on the Windows NT operating system platform.

1.0 Introduction

The Anti-debugging concept evolved as a technique by software vendors to hide their code, in other words, to stop reversing their application, and thus not exploiting the vulnerabilities. This idea was taken over by the malware writers and they implemented such anti-debugging code into their malwares to hide from AV engines and reverse engineers. Some of the anti-debugging techniques involve detection of debugger control, execution inside a virtual environment or code being emulated by a tracer application. All techniques discussed here vary from simple techniques like registry entries to ring 3 debuggers like Olly, Immunity, and IDA to the ring 0 debuggers like softICE and Windbg. For ease of understanding, I have inserted tested VC2005 snippets for illustration. All tricks are tested under windows XPSP2 unless specified.

Specific Debugger detection techniques

2.0 Olly Debugger Detection techniques

2.1. Crash the Olly with OutputDebugString()¹

The API OutputDebugString is a technique used to exploit the buffer overflow unfixed vulnerability in Olly v1.10. When a large string is purposefully sent as the parameter to the API, the Olly Crashes. This method has been implemented in Themida packer. The code snippet can be illustrated as:

```
int crash_olly(void)
{
    OutputDebugString("%s%s%s%s%s");
}
```

2.2 Olly Prefix² Method

Olly is not perfect at handling ASM prefixes, especially prefixes with 1 byte instructions like INT1 or INT3. This can be exploited by inserting prefix In case of a normal execution exception is generated here and the code jumps to the

exception pointer, but exception doesn't occur in Olly Debugger. (Single Step break point is otherwise called as ICE breakpoint.) This method can be illustrated in ASM as:

```
PUSH offset SehContinue
PUSH DWORD PTR FS:[0]
MOV DWORD PTR FS:[0],ESP
Prefix: 0xF1 ;The 1 byte Prefix Instruction. 0xF1:Int1
POP DWORD PTR FS:[0]
ADD ESP,4
POPAD
PUSH @Debugger_Detected
CALL EXIT_PROCESS
SehContinue:
PUSH @Debugger_Not_Detected
```

2.3 Olly Memory PAGE_GUARD Handler³

OllyDBG interprets PAGE_GUARD⁴ as a Memory break-point. If we set SEH and execute a PAGE_GUARDED code, exception will occur. If debugger is present it will execute the debugger handler for handling memory breakpoints and continue executing its code after it. If debugger is not present handling will be forwarded to SEH. A time based attack can be topped up to this technique too. The disadvantage is that the exception is generated only once. The PAGE_GUARD has to be set up again if required again for the next Exception generation.

2.4 Olly ESI Technique⁵

Olly breaks at the program Entry point with a value of 0xFFFFFFFF in ESI. This technique can be made use of to find the Olly Debugger. However, This technique has been tested and proven to be vulnerable in Windows XP SP1 only.

3.0 SoftICE Detection techniques⁶

3.1 SoftICE Driver Detection thro CreateFileA()

ASProtect 2.1 detects the presence of debugger SoftICE by trying to create a handle to the softICE driver by calling the API CreateFileA() with "\\.\NTICE" as the filename. A handle to the created file will be returned if the debugger exists. This can be illustrated as

```
int NTICE_Detect(void) {
    if(CreateFile("\\.\NTICE", 0xE0000000);
        return 1; //debugger detected
    else
        return 0; //Debugger not detected
}
```

3.2 SoftICE Driver Detection through EnumDeviceDrivers()⁷

Enumerating through all the devices drivers loaded in the memory returns image bases of all the loaded drivers, with which the imagename can be compared with SoftICE driver name "SoftICE".

4.0 Generic debugger detection⁸ techniques

The generic debugger detection techniques are the techniques which will be functional in all kinds of generic process debugging.

4.1 IsDebuggerPresent()⁹ API Method

OS Version: All WinNT.

This technique involves the use of the API IsDebuggerPresent(). The API returns boolean 0 if debugger is absent. This can be illustrated in C as:

```
if (IsDebuggerPresent())
    printf("Debugger Detected.");
else
    printf("Debugger is not detected");
}
```

4.2 PEB Debugger Check

The debugger bit is set to 1 in the PEB when any process is loaded by a debugger. IsDebuggerPresent() internally uses this logic to find presence of debugger.

```
7C813123 > 64:A1 18000000  MOV EAX,DWORD PTR FS:[18]
7C813129    8B40 30             MOV EAX,DWORD PTR DS:[EAX+30]
7C81312C    0FB640 02          MOVZX EAX,BYTE PTR DS:[EAX+2]
7C813130    C3                RETN
```

4.3 csrss.exe module debug method

This technique involves in opening the csrss.exe process using the API OpenProcess() in PROCESS_ALL_ACCESS mode. If this is done from a debugger, error is returned, else, handle to the Opened Process is obtained. The drawback of this technique is to determine the Process ID of "csrss.exe". This implementation can be done in C as

```
int csrss(void)
{
    HANDLE Csrss = 0;
    Csrss = OpenProcess(PROCESS_ALL_ACCESS, FALSE, PID_OF_CSRSS);
    if (Csrss != NULL) {
        CloseHandle(Csrss);
        printf("In Debugger");
        return true;
    }
    else {
        printf("Not In Debugger");
        return false;
    }
}
```

4.4 OutputDebugString()¹⁰ method

OutputDebugString() return values can be validated to check for debugger. This API is used to send stream of text data to the debugger. If successful, 1 is returned else, 0 is returned. This can be illustrated in C as

```
int odebugstr() {
    int i;
    OutputDebugString("Hello World");
    __asm
        mov i, eax;
    if(1==i)
        printf("No Debugger");
    else
        printf("Debugger");
    return 0;
}
```

Internally, the API calls DbgPrint() API which can also be used similarly for debugger validation.

4.5 BlockInput()¹¹

This API blocks the input devices of the operating system. Neither mouse nor keyboard can be used if this is set. Yoda protector uses this technique while decrypting the code. After its decryption is done, the BlockInput is called and the mouse functions are enabled.

4.6 GetTickCount()¹²

This technique uses calling API twice with some arbitrary time calculated code between the two calls. If the time obtained is greater than the time calculated arbitrary code to execute then we can confirm the presence of a debugger.

```
BOOL tick_method(void) {
    DWORD a, b, c;
    a=GetTickCount();
    for(i=0;i<0xFFFF;i++)
        __asm nop;
    b=GetTickCount();
    if((a-b)>65555)
        printf("Debugger Detected");
    else
        printf("Debugger Not Detected");
}
```

4.7 CloseHandle()and NTClose()

NTClose() or its wrapper function CloseHandle() API when called with an invalid handle, returns STATUS_INVALID_HANDLE (0xC0000008) exception when being debugged. This exception was of late seen in the malware family "Trojan-Downloader.Win32.Zlob" called this function repeatedly in its unpacking loop.

4.8 INT3 or 0xCC instruction

Any debugger in user mode, when a breakpoint is set at a point means that a byte instruction 0xCC is set in between the two lines of code. 0xCC

corresponds to the INT3 instruction. The code which when is made to run, will intercept the byte 0xCC and the processor generates an exception. This exception is caught by the debugger and the process is suspended in the background. This is taken advantage of and INT3 instructions along with SEHandlers containing the resume code are inserted purposefully into the code. Now, when an INT3 is intercepted, the debugger's SEHandler is called but not our written SEHandler making the process to exit. This is an old school method and many packers from yoda to AsProtect use this technique. This can be illustrated as

```

push offset @SEHandler
push dword fs:[0]
mov fs:[0], esp

INT3

@SEHandler
Continue Execution

```

4.9 Interrupt 2Dh

Executing this interrupt if the program is not debugged will raise a breakpoint exception. In case of the code being debugged with the trace flag, the instruction is not executed so no exception will be generated. Setting up this interrupt is similar to INT3. This can be found in the recent malware families like "NetWorm.Win32.KoobFace".

Trap Flag	INT 0x2D Exception
Set	Exception Not Generated
Not Set	Exception Generated

4.10 Setting the Trap Flag bit.

Setting the value of the trap flag bit to 1 will cause a SINGLE_STEP for every instruction traced. This will cause a lot of problems to the debugger. We make use of PUSHFD and POPFD to implement this. This is illustrated above. When this code is executed normally, no exceptions are generated

```

pushf;
or [esp + 1], 0x01;
popf;

```

4.11 PUSHSS, POPSS, and PUSHF¹³

This trick makes use of the processor mechanism of keeping itself protected. In this trick, I have used POP SS, which is a protected mode instruction. When any protected mode instruction changes its value, in our case, it is POP SS; the processor will not allow any interrupts by setting the trap flag high until the next 1 instruction execution is completed. It then resets the trap flag

also. When any debugger traces on POP SS, the processor has disabled all the interrupts, so the debugger does not break at the next instruction. Here, we push the 16 bit flag register to the stack using PUSHF. Now all we have to do is to take the value pushed in the stack and check if the trap flag was set while PUSHF instruction was executed.

```
int _tmain(int argc, _TCHAR* argv[])
{
    int a=0;
    __asm
    {
        xor eax,eax;
        push SS;
        pop SS;
        pushf;
        pop AX;
        and EAX,0x100;
        or EAX, EAX;
        jz debugged;
        mov a, 0x0;
        jmp exit;
    }
    debugged:
        mov a, 0x1
    exit:
    };
    if(a)
        printf("No Debugger");
    else
        printf("Debugger Exists");
}
```

As an extension to this of my research, I did test other interrupt services like INT 0x2C, RDTSC in a Virtual Machine and found them to give the same result.

4.12 RDTSC – Time Keeping

This is a time keeping technique as similar to calling the function GetTickCount() function. In case of a VM, this RDTSC interrupt service routines are not as accurate as it is done in a host machine hence this can also be used as an Anti-Virtualization technique for VMWare.

4.13 File and Memory CRC

The famous TELock packer calculates CRC of the file and memory against modification, the check fails if any breakpoint like INT3 is encountered. This is a very powerful but CPU cycles consuming technique.

5.0 Anti-Virtualization Techniques

5.1 Detecting presence of VMWare

5.1.1 Red Pill Technique¹⁴

The redpill technique was found out in Nov 2004. It makes use of the SIDT instruction, to read the contents of the Interrupt descriptor table register. Any system can have only one IDTR register. If a VM is under execution, the VM's IDTR needs to be relocated to a newer space to avoid conflict with the host. This

is the main technique followed behind the red pill technique. Several Virtual machines either Windows or Linux or honey pots falls prey to this technique.

5.1.2 SLDT Method

This method is similar to the previous method. This makes use of SLDT instruction which can be illustrated as.

```
unsigned char m[2];
__asm sldt m;
return (m[0] != 0x00 && m[1] != 0x00) ? 1 : 0;
```

5.1.3 Using Privilege Instruction OUT and IN¹⁵

IN is a privilege instruction in the Intel instruction set. VMWare uses these instructions to communicate with the host. So by default, the instruction will be executed in a VM but the host will throw an exception. This can be illustrated as

```
MOV EAX, 564D5868h          /* magic number */
MOV EBX, command-specific-parameter
MOV CX, backdoor-command-number
MOV DX, 5658h              /* VMware I/O Port */
IN EAX, DX (or OUT DX, EAX)
```

5.2 Skipping Nod32 Emulation

We all are aware of the NOD32 emulation techniques. But NOD32 emulation techniques are not extended to MMX or 3dnow instruction. This technique is simple; just add `__asm pminsw xmm0, xmm1` to your first line of code and one can easily skip NOD32 emulation.

5.3 Detecting WINE and Honey pots

This trick uses a simple technique where it calls some unimplemented WINE API emulation calls like “ZwWriteprocessMemory” instead of “WriteProcessMemory”. In this case, the file would not be executed in a WINE environment.

5.4 Detecting sandboxes

We use different sandboxes like ThreatExpert, SandBoxie, Joebox, Anubis, CWSandbox, Norman Sandbox, CAMAS etc. They have a unique pattern of sample execution, all sample need to be executed under windows environment and the windows copy needs to be registered and would be unique. Malwares exploited this technique by reading the value from the registry key “HKLM\Software\Microsoft\Windows\CurrentVersion”, queries the value of “ProductId” from them and compares with Mal-backlisted keys like “76487-337-8429955-22614” for Anubis Sandbox, “76487-644-3177037-23510” for CWSandbox and “55274-640-2673064-23950” for Joebox.

ThreatExpert and SandBoxie uses dll's to intercept the API calls and virtualizes the process. ThreatExpert uses “DbgHelp.dll” and sandboxie uses “sbiedll.dll”. If these DLL's are loaded, use GetModuleHandle() to get its handle and check for their existence.

6.0 Conclusion

I hope this paper would have explained currently prevailing anti-debugging techniques used by different protection software and malwares. I would also say that these are not the only anti-debugging technique but new techniques evolve with different updates and patches and the role of a researcher or a forensic analyst is to keep oneself up to date with such techniques.

¹ Illustration available at: <http://www.reversing.be/forum/viewtopic.php?p=152&sid=784c4616fddd17ade9e7cd611d97c215> [Jan 16 2010]

² Implementation of prefixes can be found at <http://ap0x.jezgra.net/protectionLab/prefixes-OllyDBG.rar> [Jan 16 2010]

³ Read more on its implementation available at: <http://ap0x.jezgra.net/protectionLab/OllyPageGuard.rar> [Jan 16 2010]

⁴ Read more about setting PAGE_GUARD from Microsoft available at: [http://msdn.microsoft.com/en-us/library/aa366549\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa366549(VS.85).aspx) [Jan 16 2010]

⁵ Read more on the implementation available at: <http://ap0x.jezgra.net/protectionLab/OllyDBG-ESI-Trick.rar> [Jan 16 2010]

⁶ Forum Discussion on NTICE detection available at: <http://www.woodmann.com/forum/showthread.php?t=9688> [Jan 16 2010]

⁷ Implementation of code is available at: <http://ap0x.jezgra.net/protectionLab/antiNTIce.rar> [Jan 16 2010]

⁸ Different methodologies of detection available at: <http://www.securityfocus.com/infocus/1893> [Jan 16 2010]

⁹ Documentation of API available at: [http://msdn.microsoft.com/en-us/library/ms680345\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms680345(VS.85).aspx) [Jan 16 2010]

¹⁰ Documentation of API available at: [http://msdn.microsoft.com/en-us/library/aa363362\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/aa363362(VS.85).aspx) [Jan 16 2010]

¹¹ Documentation and usage details Available at: [http://msdn.microsoft.com/en-us/library/ms646290\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms646290(VS.85).aspx) [Jan 16 2010]

¹² Documentation available at [http://msdn.microsoft.com/en-us/library/ms724408\(VS.85\).aspx](http://msdn.microsoft.com/en-us/library/ms724408(VS.85).aspx) [Jan 16 2010]

¹³ Read more on the internals available at: <http://www.ragestorm.net/blogs/?p=45> [Jan 16 2010]

¹⁴ Read more on the red pill available at: <http://www.invisiblethings.org/papers/redpill.html> [Jan 17 2010]

¹⁵ Illustration and commands on backdoor ports available at: <http://chitchat.at.infoseek.co.jp/vmware/backdoor.html> [Jan 17 2010]