J Comput Virol (2012) 8:141–149
DOI 10.1007/s11416-012-0165-0

ORIGINAL PAPER

Anti-virtual machines and emulations

Anoirel Issa

Received: 16 February 2012 / Accepted: 1 May 2012 / Published online: 19 June 2012

Abstract Virtual Machines are important infrastructural tools for malware analysis. They provide safe yet accurate way of evaluating real life behavior and impact of any executable code, thus providing a better understanding of obfuscated or non conventional portions of code within a binary file. Many virtual machines, such as VMware, Qemu, VirtualBox and SandBoxes, are available and are widely adopted by malware researchers and analysts. Moreover, many antivirus scanners have their own implementation of emulators to achieve comparable results by running malicious code within a controlled environment in order to decrypt obfuscated code. Virus writers have always responded to these technologies. Most malware today uses anti-debug techniques to counter analysis and evade antivirus detection. Lately, malware like Zeus/SpyEye and associated families such as Smoaler, Dromedan, Kazy, Yakes, and other malware such as Spyrat or W32.Pilleuz, have deployed techniques to disrupt the use of virtual machines and emulators. These malware families are able to implement different variations of disruption techniques within single samples or within related groups of malware before propagation. This paper will present a study of these anti-emulation and anti-virtual machine techniques.

1 Introduction

In today's complex malware threats, cybercriminals invent and implement different technologies that would protect their malicious code from being reverse engineered and understood by anti-malware analysts. The first protection technologies are packers and encryptors. They are available commercially or freely on the Internet. Packer protection systems are obfuscation tools used by a wide array of software companies who wish to protect their intellectual property. Virus writers use packers a lot to obfuscate their malware before propagation. According to antivirus company data, the vast majority of malware is protected by packers. Packers are very popular because there is no development time required in order to protect specific software; in that sense they are very cost effective. However there is a downside to this technology, most packers are very well known. Many unpacking technologies are implemented in antivirus scanners and other reverse-engineering tools.

There are other protection mechanisms, such as anti-debug, used to prevent automated or human analyzers from accessing the core functionality of the malicious code. However similar to the packer technology, most anti-debug techniques are very well know and there are lots of publicly available resources and documentation about them.

There is now emulation and anti-emulation technology. Although this is not a new technology, implementing anti-emulation techniques requires more skill that most of the previously cited methods of protection. Cyber criminals have understood that virtual machines and emulators are the safest environment used to analyze and evaluate their malicious code. Some professional malware writers such as those responsible of developing Zeus or Spyrat have decided to focus on developing and implement as many anti-emulation technologies as possible in order to disrupt analysis attempts on their code. With a lack of innovative technologies they recycle already-existing concepts and turn them into subtle, but fairly new techniques, thus enabling malware that can detect when they are running in a hostile environment.

A. Issa (✉)
Security Technology and Response Division,
The Global Malware Services Department, Symantec,
1260 Lansdowne court, Gloucester Business Park,
GL3 4AB Gloucester, UK
e-mail: anoirel_issa@symantec.com

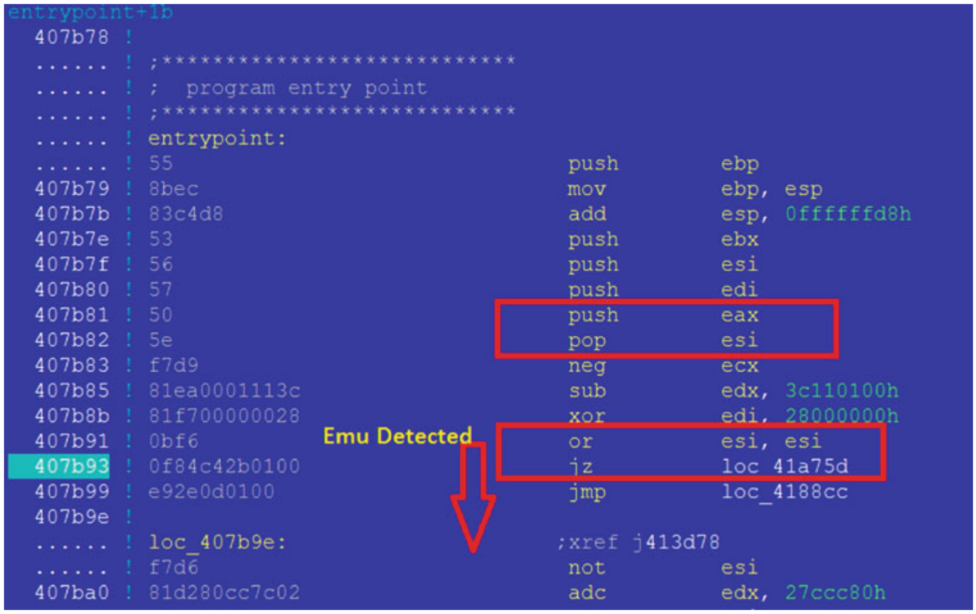

Fig. 1 CPU register-based anti-emulator under Windows 7 and Vista

2 Central processing unit (CPU) registers based anti-emulation

When a program is executed, the operating system initializes its environment first. Specific memory regions such as the stack and heap are allocated and reserved so that the program can use them in order to carry out its task. Many of these environmental settings are predictable. Although their predictability has significantly been reduced by the introduction of the address space layout randomization (ASLR), it is still possible to predict some other variables within a program's environment. For example, the initial values of the CPU registers can be known prior to the program's execution.

Each emulation system, such as virtual machines and emulators, can present their own initial register characteristics that are different than those in a non-emulated environment. For instance VirtualBox, which tends to be targeted by many malware, will have different environmental settings than the Pokas emulator, an open source emulator. This also means that the emulator in antivirus program A is likely to differ from that of antivirus program B.

By checking the state and initial values of these registers at the entry point, the malware can deduce whether or not it is being analyzed in a virtual environment, even which emulator is analyzing it. This technique has been heavily used by Zeus and the like since November 2011 to detect virtual machines and emulators.

3 Targeting Windows 7 and Vista

Here is an example of CPU register-based anti-emulation for ASLR based systems such as Windows 7 and Vista.

Virus Name: Zbot

On newer operating systems that have address space layout randomization (ASLR) implemented, some registers are expected not to have some specific values. For instance, the EAX register shouldn't be zero. This has been exploited by some malware in order to check the environment it is running within. Assuming that some emulators will initialize registers at the entry point to zero, an efficient attack against these emulators is to check for a value that can be predicted. For instance, the value of the EAX register.

In the Fig. 1 above, the initial value of the EAX register is tested to decide whether emulation is present. In the event of a different value than what is expected, the program is certain that it is run in a non-conventional environment such as an emulator.

Figure 2 illustrates the destination of the code in Fig. 1 refers to "jz loc_41a75d".

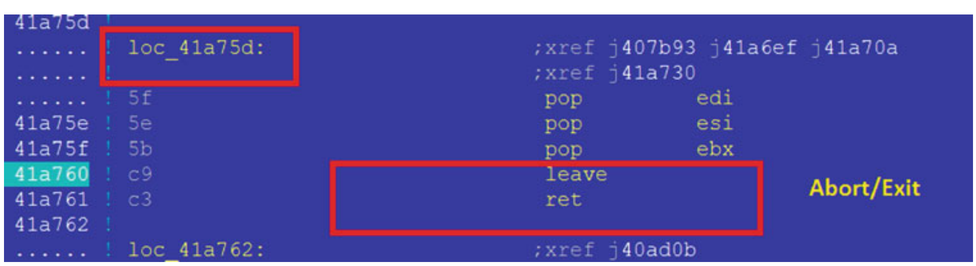

Fig. 2 Exit program. Emulation is detected

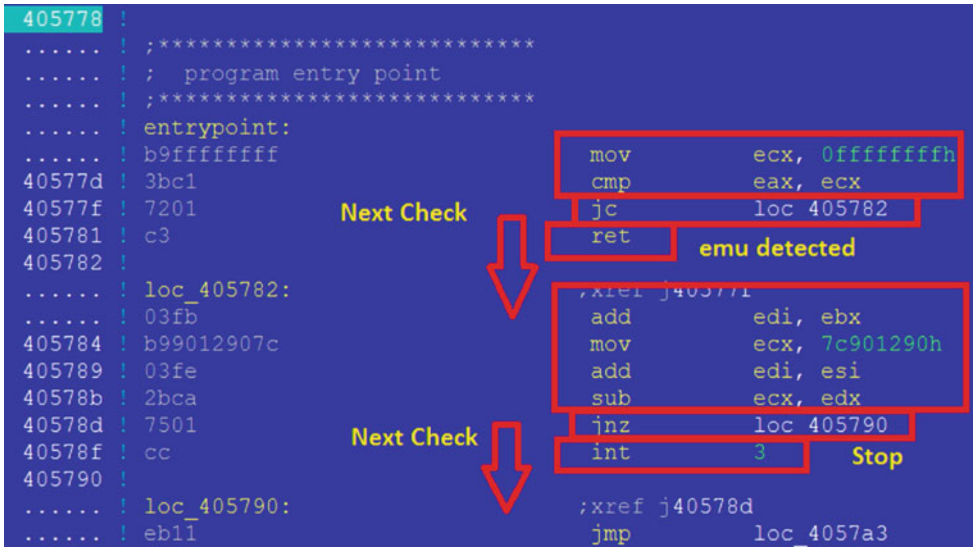

Fig. 3 A more universal anti-emulation

The code in the Fig. 2 is a program-termination routine. Once the malware has detected the presence of the emulation, it simply stops executing itself and exits.

4 Different samples targeting different systems

In non-ASLR systems like Windows XP, the EAX entry point is always zero, so this detection process would not work. However the botnet responsible in spreading Zeus can send hundreds of samples per month. Sometimes the samples are the same, except that the anti-emulation, anti-debug or anti-virtual machines used are different. This gives them the flexibility of targeting different systems with their new anti-analysis code.

5 A more 'universal' approach

The first code we've seen was targeting Windows Vista and subsequent versions with ASLR enabled such as Windows 7. However malware can also target just about any system. The following code is another implementation that would work on Windows XP, Vista, and Windows 7.

Figure 3 presents an anti-emulation technique that should work from either Widows XP or Vista. The approach is to check the value of the ECX register as part of the first anti-emulation check. Under Windows XP, ECX should always point to an address within the stack range when a program starts; however, under Windows 7 ECX starts with a value of zero. Therefore, if the value of that register is 0xffffffff the least we can say is that the environment is not conventional. Since many tools initialize values with either 0 or 0xffffffff, the malware can deduct that it is being run in a hostile environment.

6 Stack address range anti-emulation

Just as the initial values of the CPU registers can be known prior to a program is executed, the stack address range can be known in advance. A stack-based, anti-emulation technique has been implemented by the same malware family. It consists in checking the address range of the stack against the running process. When executed, the stack allocated to processes under Windows XP can be predicted. By checking the address range, it is possible to determine if a program is run under emulation.

Figure 4 illustrates the initial registers state on a Windows XP computer.

As highlighted on the above illustration, the ECX and ESP registers point to values in the stack. Under Windows XP, or a similar environment, the stack address range is usually based around 0x120000. In Windows 7 the stack seems to be based around 0x180000. So by checking the stack address range, malware can determine whether it is running under an emulated environment and subsequently abort its actions and exit or trigger a system crash.

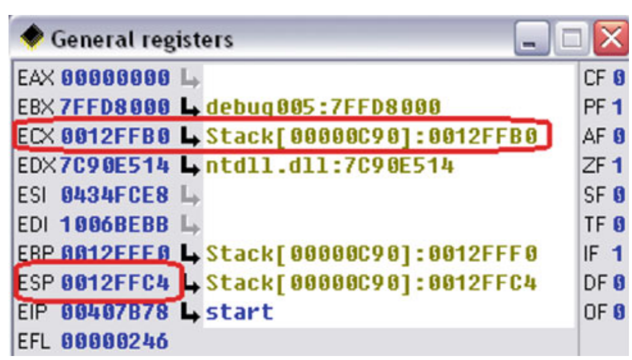

Fig. 4 Stack address range

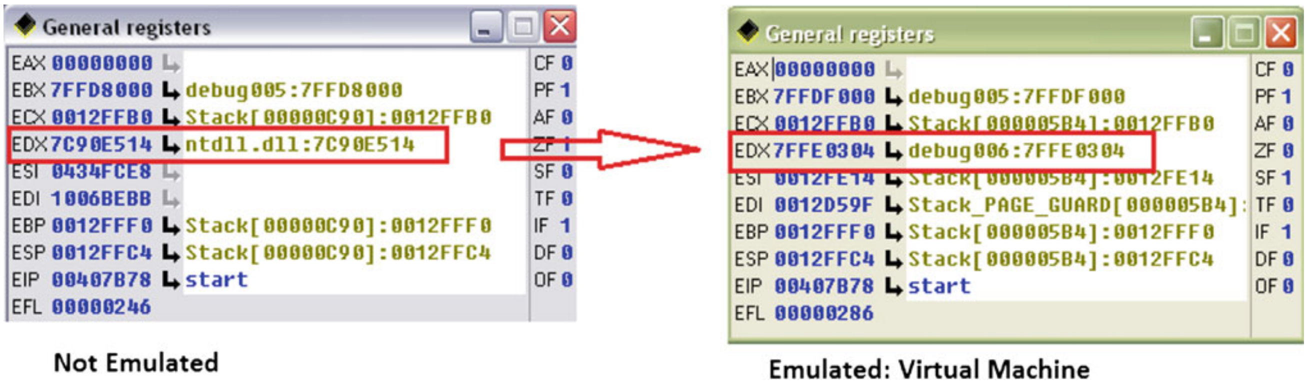

Fig. 5 EDX values in non-emulated and virtual environments

7 Dynamic linked library address space checks as anti-emulations

Dynamic linked libraries are often needed in programs, including malware. One of the most commonly used DLLs in a Windows environment is kernel32.dll. DLLs are mapped in memory regions that are very predictable in a non-ASLR-enabled environment. Knowing the memory address range of a specific DLL, malware is able to determine if they are being analyzed in an emulated system by checking the address range of some libraries. The idea behind this technique is similar to the one that checks the stack address range.

This time, what is checked is the address range of a particular DLL. For instance, it is known that upon execution, the entry point the value of the EDX register points to is an address within Ntdll.dll, which is around 0x7C90000 in Windows XP, as shown in the Fig. 5.

Figure 5 shows this under a normal, non-emulated environment. On a system running Windows XP, the EDX register points to the ntdll.dll address of 0x7c90E514, which is the address of the KiFastSystemCallRet system API. This is the expected behavior. However under a virtual machine, (for instance VirtualBox) EDX no longer points to an address within ntdll.dll, but to a debug register address. So by checking whether EDX points to the address range of ntdll.dll at the entry point, malware can detect if it is being analyzed in a virtual machine or emulator.

8 Junk APIs as anti-emulation

Although this technique has been heavily used over a year now, it continues to be used albeit to a lesser extent. A lot of malware abusively uses junk API calls, whether with illegal parameters or with valid ones, several times so that they can break some emulators.

This is a direct attack on emulators that don't handle API calls very well, since in the case of non-handled APIs, it may not execute the malware in its virtual environment, classifying the threat as "clean" by ignoring it.

9 From simplicity to complexity: exploiting the CPU based anti-emulation and virtual machines

From the simplicity of the initial values of the CPU registers, there are a certain number of ways anti-emulation can be implemented. Different samples can have variations that can be simple, but also very complex. Malware that implements these anti-emulation techniques, coupled with obfuscation such as polymorphism, can be very difficult to track down.

10 Statistics on one CPU-based, anti-emulation malware family

The Fig. 6 is a chart showing the statistics on a malware family that uses one of the CPU-based, anti-emulation techniques over a period of six months. Although this particular technique has not been used lately, it has been seen in over 34,000 times during the period.

11 Anti-Sandboxes

Sandboxes are tools that offer sandboxed virtual environment to run programs and observe their behavior. As a result, these tools, popular among the reverse-engineering community, are also under close scrutiny by the malware-writing community.

12 Anti-Sandboxie

As most of the programs have got their own sets of files, the Sandboxie sandbox has a core dynamic link library called 'SbieDll.dll'. As illustrated above, some malware checks if this particular DLL has been loaded in the system. If so,

	Last Hour	Last 12-Hours	Last Day	Last Week	Last 2-Weeks	Last Month	Last 3-Months	Last 6-Months
	0	0	0	0	0	0	0	34,069
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	34,069

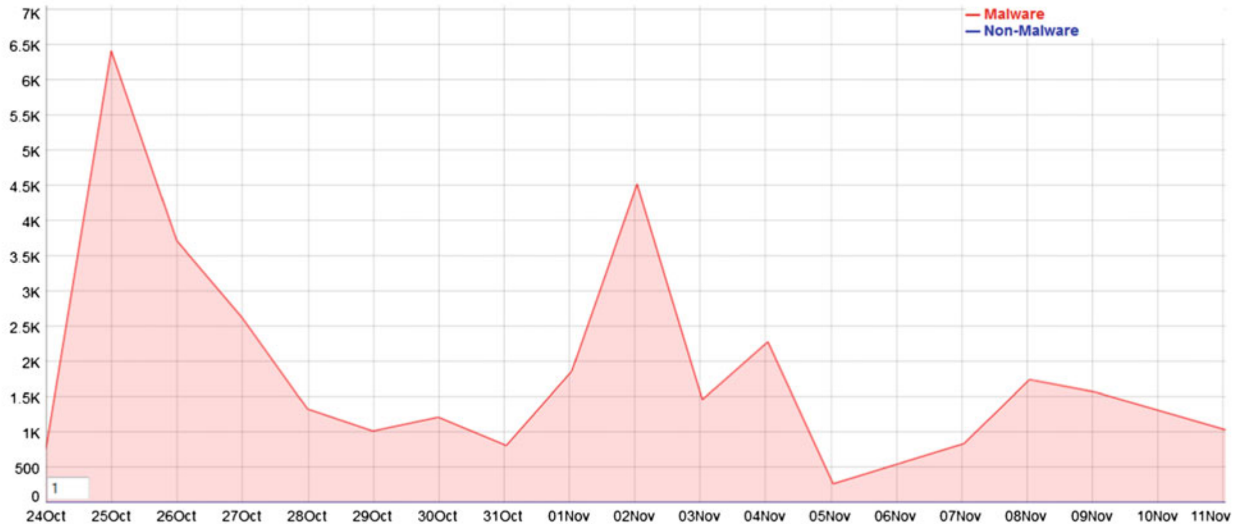

Fig. 6 Statistics on a malware family using a cpu based anti-emulation

```
CODE:004052EC
CODE:004052EC SandBoxie_Check_SbieDll_dll proc near    ; CODE XREF: ANTI_DEBUGGER_
CODE:004052EC                                          ; DATA XREF: ANTI_DEBUGGER_
CODE:004052EC                push    ebx
CODE:004052ED                xor     ebx, ebx
CODE:004052EF                push    offset aSbiedll_dll ; "SbieDll.dll"
CODE:004052F4                call    GetModuleHandleA_0
CODE:004052F9                test    eax, eax
CODE:004052FB                jz      short loc_4052FF
CODE:004052FD                mov     bl, 1
CODE:004052FF
CODE:004052FF loc_4052FF:                              ; CODE XREF: SandBoxie_Chec
CODE:004052FF                mov     eax, ebx
CODE:00405301                pop     ebx
CODE:00405302                retn
CODE:00405302 SandBoxie_Check_SbieDll_dll endp
CODE:00405302
```

Fig. 7 Anti-Sandboxie

then it is certain that the malware is in fact running under a sandboxie emulated environment Fig. 7.

13 Anti-VMware

VMware is one of the most popular virtual machines available. It offers an easy interface to work with and supports a wide range of operating systems such as DOS, OS2, Linux, and Windows. It is naturally one of the earliest to be targeted by malware. The technique here is rather common but still widely used by malware wishing to detect VMware. Figure 8 represents code recently seen in a Spyrat sample.

VMware uses the EBX number 564D5868h, which corresponds to the ASCII value ‘VMXh’, along with communication port 5658h, which has a corresponding ASCII value of ‘VX’. The command 0ah returns the VMware version. One of the most common ways of detecting VMware is to issue

```
CODE:00405124 arg_8          = dword ptr  0Ch
CODE:00405124
CODE:00405124                xor     eax, eax
CODE:00405126                push    offset loc_40514C
CODE:0040512B                push    dword ptr fs:[eax]
CODE:0040512E                mov     fs:[eax], esp
CODE:00405131                mov     eax, 'VMXh'     ;  Vmaware magic
CODE:00405136                mov     ebx, 3C6CF712h
CODE:0040513B                mov     ecx, 0Ah        ; version
CODE:00405140                mov     dx, 'VX'        ; vmware port
CODE:00405144                in      eax, dx         ; read the port
CODE:00405145                mov     eax, 1
CODE:0040514A                jmp     short loc_40515F
CODE:0040514C ; ---------------------------------------------------------------
CODE:0040514C
CODE:0040514C loc_40514C:                            ; DATA XREF: Anti_VMWARE+2↑o
CODE:0040514C                mov     eax, [esp+arg_8]
CODE:00405150                mov     dword ptr [eax+0B8h], offset loc_40515D
CODE:0040515A                xor     eax, eax
CODE:0040515C                retn
```

Fig. 8 VMware detection

```
CODE:0040523D         call    convert_String_to_UpperCase
CODE:00405242         mov     eax, [ebp+var_12C]
CODE:00405248         push    eax
CODE:00405249         lea     edx, [ebp+var_138]
CODE:0040524F         mov     eax, offset aVboxservice_ex ; "VBoxService.exe"
CODE:00405254         call    convert_String_to_UppaVboxservice_ex db 'VBoxService.exe',0
CODE:00405259         mov     eax, [ebp+var_138]
```

Fig. 9 Anti-VirtualBox

a VMware command and check if the return value in EBX (which should differ from the original value) is ‘VMXh’. If this is the case then VMware is present.

The other way is to set EAX to a value of ‘1’, like in the example in the Fig. 8, then define exception-handing code that clears the EAX register if it gets executed. If an exception occurs when running the VMware commands code, then EAX should not be ‘1’ anymore but ‘0’.

This way by checking the value of EAX, it is possible to determine whether VMware is running or not.

14 Anti-VirtualBox

VirtualBox is an open source virtual machine owned by Oracle. Like VMware, VirtualBox is a popular tool. Malware writers have found a simple way of detecting its presence, as illustrated below in Fig. 9.

Like many other tools, VirtualBox has its own running processes. As shown in the code above, taken from a recent malware sample, the simplest way to detect the presence of VirtualBox is to scan all running processes and check for VboxServices.exe. The figure below shows a list of VirtualBox processes on a system Fig. 10.

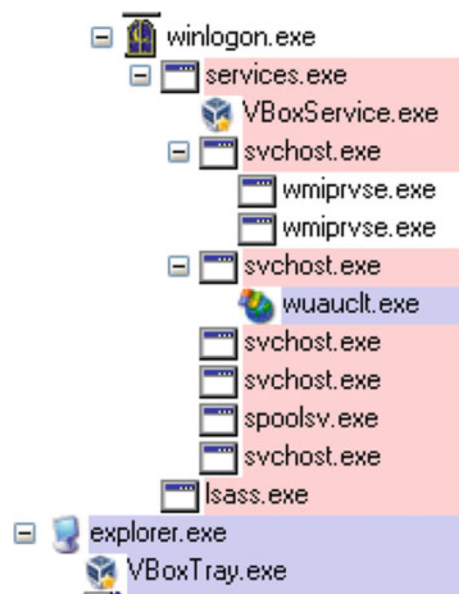

Fig. 10 VirtualBox processes

VirtualBox can also be detected by checking for the VboxTray.exe process. However, if the user didn’t install additional VirtualBox tools, then this is not always present.

```
CODE:004054B0                push    0                   ; ulOptions
CODE:004054B2                push    offset aSoftwareMicr_1 ; "Software\\Microsoft\\Windows\\
CODE:004054B7                push    80000002h           ; hKey
CODE:004054BC                call    RegOpenKeyExA
CODE:004054C1                test    eax, eax
CODE:004054C3                jnz     short loc_4054F7
CODE:004054C5                mov     [esp+110h+cbData], 101h
CODE:004054CD                lea     eax, [esp+110h+cbData]
CODE:004054D1                push    eax                 ; lpcbData
CODE:004054D2                lea     eax, [esp+114h+Data]
CODE:004054D6                push    eax                 ; lpData
CODE:004054D7                push    0                   ; lpType
CODE:004054D9                push    0                   ; lpReserved
CODE:004054DB                push    offset aProductid ; "ProductId"
CODE:004054E0                mov     eax, [esp+124h+hKey]
CODE:004054E4                push    eax                 ; hKey
CODE:004054E5                call    RegQueryValueExA
CODE:004054EA                lea     eax, [esp+110h+Data]
CODE:004054EE                cmp     eax, offset asc_405544 ; "76487-337-8429955-22614"
CODE:004054F3                jnz     short loc_4054F7
CODE:004054F5                mov     bl, 1
CODE:004054F7
CODE:004054F7 loc_4054F7:                                ; CODE XREF: Anti_Anubis_Sandox+1F↑j
```

Fig. 11 Anubis detection by product ID

```
CODE:0040541E                push    eax                 ; lpData
CODE:0040541F                push    0                   ; lpType
CODE:00405421                push    0                   ; lpReserved
CODE:00405423                push    offset aProductid_0 ; "ProductId"
CODE:00405428                mov     eax, [esp+124h+hKey]
CODE:0040542C                push    eax                 ; hKey
CODE:0040542D                call    RegQueryValueExA
CODE:00405432                lea     eax, [esp+110h+Data]
CODE:00405436                cmp     eax, offset a76487644317703 ; "76487-644-3177037-23510"
CODE:0040543B                jnz     short loc_40543F
CODE:0040543D                mov     bl, 1
CODE:0040543F
CODE:0040543F loc_40543F:                                ; CODE XREF: Anti_CWSandBox+1F↑j
CODE:0040543F                                            ; Anti_CWSandBox+4F↑j
```

Fig. 12 CWSandbox detection

15 Anti-Anubis SandBox

Anubis is an online malware analysis sandbox. It is useful when one wants to quickly check a file's behavior. Although this is not a standalone application, accessible directly by the customers, malware writers have managed to get some of Anubis's environmental settings, such as the product ID.

It is known that Anubis uses the following key as its product ID: 76487-337-8429955-22614.

Figure 11 illustrates how malware checks for Anubis.

By checking for the presence of this particular value in the registry, malware is able to detect whether it is running in an Anubis sandbox.

16 Anti-GFI CWSandbox

CWSandbox is accurately described by its owner as an automated malware analysis tool. It is an advanced tool that offers lots of interesting features. Malware writers discovered that this tool uses a constant product ID, so they use it as a way to detect its presence. The figure illustrates a CWSandbox detection Fig. 12.

The same technique used to detect Anubis is applied to the detection of CWSandbox. The malware reads the registry and tries to locate the following product ID associated with CWSandbox: 76487-644-3177037-23510. If this value is present then the malware knows that this it is being analyzed under a virtual environment.

```
CODE:00405367                 push    0               ; lpType
CODE:00405369                 push    0               ; lpReserved
CODE:0040536B                 push    offset ValueName ; "ProductId"
CODE:00405370                 mov     eax, [esp+124h+hKey]
CODE:00405374                 push    eax             ; hKey
CODE:00405375                 call    RegQueryValueExA
CODE:0040537A                 lea     eax, [esp+110h+Data]
CODE:0040537E                 cmp     eax, offset a55274640267306 ; "55274-640-2673064-23950"
CODE:00405383                 jnz     short loc_405387
CODE:00405385                 mov     bl, 1
CODE:00405387
CODE:00405387 loc_405387:                             ; CODE XREF: Anti_JoeBox_SandBox+1F↑j
```

Fig. 13 JoeBox detection

17 Anti-JoeBox Sandbox

Joe Box is yet another sandbox for behavioral analysis. Like some of the other sandboxes, its product is known and is used in return to detect it. The figure below shows how it is detected by a malware sample Fig. 13.

Once again, to detect the presence of the JoeBox sandbox, the malware scans the registry key for the presence of the following product ID: 55274-640-2673064-23950. If this value is found then JoeBox is likely running, and the malware stops its operations.

18 Anti-Debug help library

The debug help library dbghelp.dll is a DLL provided by Microsoft to support debugging of executable binary files in Windows. Many tools use this library for debugging. Malicious software can check if this **dbghelp.dll** library is loaded in memory in order to detect if they are running under a debugger or similar environment. Figure 14 below shows how a malware sample is checking for the presence of this particular DLL.

This code shows how malware can check if dbghelp.dll is loaded into memory. The interesting part here is that this same DLL name is used by Olly Debugger for its main DLL.

19 Anti-Norman SandBox

Norman sandbox is one of the oldest professional sandboxes available. It is well known by malware authors since the early ages. However, it seems that recent malware uses the same old technique to define whether they are run under this sandbox. The figure below illustrates how the Norman sandbox is detected Fig. 15.

It is known that the Norman sandbox can be detected by querying the CurrentUser username in the registry. An old trick, but still used today.

```
CODE:00405310                                         ; DATA XREF: ANTI_DE
CODE:00405310                 push    ebx
CODE:00405311                 xor     ebx, ebx        ;
CODE:00405313                 push    offset aDbghelp_dll ; "dbghelp.dll"
CODE:00405318                 call    GetModuleHandleA_0
CODE:0040531D                 test    eax, eax
CODE:0040531F                 jz      short loc_405323
CODE:00405321                 mov     bl, 1
CODE:00405323
CODE:00405323 loc_405323:                             ; CODE XREF: Anti_Ol
```

Fig. 14 Dbghelp.dll detection

```
CODE:004055D2                 push    eax
CODE:004055D3                 lea     edx, [ebp+var_10]
CODE:004055D6                 mov     eax, offset aCurrentuser ; "CurrentUser"
CODE:004055DB                 call    convert_String_to_UpperCase
CODE:004055E0                 mov     edx, [ebp+var_10]
```

Fig. 15 Norman sandbox

```
CODE:00405814 Softice_Check_1 proc near            ; CODE XREF: Anti_Softic
CODE:00405814                 push    ebx
CODE:00405815                 xor     ebx, ebx
CODE:00405817                 push    0                ; hTemplateFile
CODE:00405819                 push    80h              ; dwFlagsAndAttributes
CODE:0040581E                 push    3                ; dwCreationDisposition
CODE:00405820                 push    0                ; lpSecurityAttributes
CODE:00405822                 push    3                ; dwShareMode
CODE:00405824                 push    0C0000000h       ; dwDesiredAccess
CODE:00405829                 push    offset FileName ; "\\\\.\\SICE"
CODE:0040582E                 call    CreateFileA_0
CODE:00405833                 cmp     eax, 0FFFFFFFFh
CODE:00405836                 jz      short Softice_Not_Installed
CODE:00405838                 push    eax              ; hObject
CODE:00405839                 call    CloseHandle
CODE:0040583E                 mov     bl, 1
CODE:00405840
CODE:00405840 Softice_Not_Installed:                  ; CODE XREF: Softice_Che
CODE:00405840                 mov     eax, ebx
CODE:00405842                 pop     ebx
CODE:00405843                 retn
CODE:00405843 Softice_Check_1 endp
```

Fig. 16 Softice detection

20 Anti-Softice

Softice is not really a virtual machine or an emulator. It is a kernel mode debugger that has been around since the DOS days. It was surprising to find recent malware that has implemented an anti-Softice technique; in fact it is quite exceptional. The figure shows how the sample is checking for Softice Fig. 16.

To detect Softice the malware tries to open the following file names, which belong to Softice:

\\\\.**NTICE** and \\\\.**SICE**

21 Conclusion

Emulators and virtual machines present some very powerful tools to safely analyze malicious code, especially in an era of heavy obfuscation. They are one of the last solutions to stand when most of the other types of analyzers have failed. It seems that this fact is taken very seriously by professional malware authors, who are investing a lot of time and energy to avoid these environments, whether it is for innovating, reinventing or recycling their own code, or using preexisting techniques. The recent focus on emulation systems makes virtualization one of the biggest trends in modern computing. The need to have an emulator that is maintained over time to tackle existing and emerging anti-emulation technologies seems to be slowly becoming a requirement.

References

1. Wikipedia, CPU Emulator http://en.wikipedia.org/wiki/Emulator#CPU_simulator
2. Wikipedia, Virtual Machine http://en.wikipedia.org/wiki/Virtual_machine
3. Symantec, Trojan.Zbot http://www.symantec.com/security_response/writeup.jsp?docid=2010-011016-3514-99
4. Symantec, W32.Spyrat http://www.symantec.com/security_response/writeup.jsp?docid=2010-011211-1602-99