Methods for Virtual Machine Detection

Alfredo Andrés Omella
Grupo S21sec Gestión S.A.
http://www.s21sec.com

20th June 2006

Abstract

This document analyses the most important methods to detect the existence of virtual machines in an execution environment, focusing on VMware virtualization fotware [1]. At the same time, it presents another way to detect the presence of this type of software using a minimal set of instructions in assembler.

1 Introduction

Nowadays, from the point of view of security (viruses, trojans, malware in general, etc.), the importance of virtual machines is growing increasingly. To be able to analyse malicious code in a virtual machine environment, makes the work of the Analyst easier. Naturally, malware programmers try to make this work more difficult, detecting the existence of this type of environments in different ways. Some of these will be explained.

2 Looking for similarities

A simple way to detect VMware environments is to look for patterns in the system; for example the MAC address of the network card (it can begin with 00-05-69, 00-0C-29 o 00-50-56), specific hardware controllers, BIOS, graphic card, copyrights, Windows registry values, thre presence of the VMware Tools process in memory, etc.

Tobias Klein's tool *scoopy-doo* *[2]* includes a small Visual Basic script that looks for certain keys within the Windows registry.

Using a different approach, Kostya Kortchinsky *[3]* has presented a set of patches that modify the data within the binary file of VMware.

3 Make it safely

Ken Kato *[4]* discovered other way to detect VMware using the so called *Backdoor I/O port*. VMware uses the I/O port 0x5658 (in ASCII 'VX') to communicate with the virtual machine. It is obvious this port is not real. The verification is simple:

1. The magic number 0x564D5868 (in ASCII 'VMXh') is loaded in the EAX register.

2. The proper parameter of the command that is to be sent is loaded in EBX register.

3. The command to be used is loaded in the ECX register. For example, the command 0x0A, which brings back the VMWare version.

4. It is read from 'VX' port. If we have 'VMXh' in the EBX register, this means that we are under VMware.

There are more commands supported by the *Backdoor I/O port*; for example to obtain data from the Windows clipboard or the speed in MHz of the microprocessor. For the complete list, see note [4].

'Jerry' the tool from Tobias Klein [5] implements this type of detection.

Kostya Kortchinsky's patches we have mentioned before [3], also allow modification of the magic number 'VMXh', so that requests sent to this port without the magic number are unsuccessful.

4 One instruction to rule them all[1]

The detection mechanisms described in the previous sections present deficiencies when they become integrated in malicious software; for example shellcodes. This is due to the size of the routines in assembler code.

A more efficient method of detection is the use of a minimal set of instructions in assembler (just one).

John Scott Robin and Cynthia E. Irvine carried out a VMM (Virtual Machine Monitor) research on the Pentium platform [6]. In their research they establish a series of requirements to be fulfilled by a VMM. They report a set of instructions susceptible to misuse because they do not comply with the established requirements.

All these instructions have in common that they are system instructions. These allow, among other things, accessing registers to control or debug etc. Most of these instructions can only be executed by the operating system, but there are some that can be performed from the user's environment. These last instructions will be explanied below.

4.1 SIDT. SGDT. SLDT.

SIDT Instruction (Store Interrupt Descriptor Table) stores the content of the IDTR (Interrupt Descriptor Table Register) register, which in fact, is a selector that points into the Interrupt Descriptor Table. The instruction SGDT (Store Global Descriptor Table) stores the register value of GDTR, which is a selector that points into the global descriptor table.

SLDT instruction (Store Local Descriptor Table) stores the register value LDTR. This register is a selector that points into the local descriptor table (LDT).

[1]J.R.R Tolkien. The Lord of the Rings

There is only one IDTR register, one GDTR register and one LDTR register, but there are two operating systems being executed: the one of the virtual machine and the one of the native machine. So the VMM needs to change the localization of the different tables. This generates an inconsistency between the values of the registers in the virtual machine and the values in the native machine.

The detection mechanisms with SGDT, SIDT and SLDT instructions were implemented by Tobias Klein in his tool *scoopy-doo* [2].

The SIDT detection was also indicated by Joanna Rutkowska [7].

However it is necessary to stress that, according to the research performed by Danny Quist and Val Smith [8], if the VMM is being executed on multiprocessing hardware, the detections performed verifying the interrupt descriptor table (IDT) can fail, because there is an IDT for each one of the microprocessors.

4.2 STR. The New Kid in Town[2]

According to Intel [9] the instruction STR (Store Task Register) stores the selector segment of the TR register (Task Register) in the specified operand (memory or other general purpose register).

Which functionality has this instruction? Well, all x86 processors can manage tasks in the same way as an operating system would do it. That is, keeping the task state and recovering it when that task is executed again. All the states of a task are kept in its TSS; there is one TSS per task. How can we know which is the TSS associated to the execution task? Using STR instruction, due to the fact that the selector segment that was brought back points into the TSS of the present task. To see more deeply how x86 architecture tasks are managed, you can red the article by Jim Turley [10].

In all the tests that were done, the value brought back by STR from within a virtual machine was different to the obtained from a native system, so apparently, it can be used as a another mechanism of a unique instruction in assembler to detect virtual machines.

In the Appendix it is attached a minimal code in C with inline assembler to prove the detection.

4.2.1 STR. Tests performed

The test battery was performed with several native operating systems and with weveral virtual machines; the VMM in all cases was VMware workstation 5.5.1 bould-19175.

- Native OS: Windows XP SP2 single processor. VM OS: Windows XP SP2 and Windows 2000 Advanced Server SP4
- Native OS: Gentoo 2.6.14-gentoo-r5 single processor. VM OS: Fedora Core 5: kern 2.6.15-1
- Native OS: Ubuntu 6.06 Dapper single processor. VM OS: Windows XP, Windows XP SP2, Windows 2000 Server SP0, Windows 2000 Server SP4, Windows Vista Ultimate Edition Version 6.0.

[2]The Eagles. Hotel California

References

[1] VMware Inc., VMware virtualization software http://www.vmware.com/

[2] Tobias Klein, scoopy doo - VMware Fingerprint Suite http://www.trapkit.de/research/vmm/scoopydoo/scoopy_doo.htm

[3] Kostya Kortchinsky, Multiple patch for VMware http://honeynet.rstack.org/tools/vmpatch.c

[4] Ken Kato, VMware Backdoor I/O Port http://chitchat.at.infoseek.co.jp/vmware/backdoor.html

[5] Tobias Klein, jerry - A(nother) VMware Fingerprinter http://www.trapkit.de/research/vmm/jerry/jerry.htm

[6] John Scott Robin, Cynthia E. Irvine, Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor http://www.cs.nps.navy.mil/people/faculty/irvine/publications/2000/VMM-usenix00-0611.pdf

[7] Joanna Rutkowska, Red Pill... or how to detect VMM using (almost) one CPU instruction http://invisiblethings.org/papers/redpill.html

[8] Danny Quist, Val Smith, Detecting the Presence of Virtual Machines Using the Local Data Table http://www.offensivecomputing.net/files/active/0/vm.pdf

[9] Intel Corporation, Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual http://www.intel.com/design/pentium/manuals/243191.htm

[10] Jim Turley, Managing Tasks on x86 Processors http://www.embedded.com/showArticle.jhtml?articleID=55301875

A Code

```
#include <stdio.h>
int main(int argc, char **argv)
{
  unsigned char mem[4] = {0,0,0,0};
  __asm str mem;
  if ( (mem[0]==0x00) && (mem[1]==0x40))
    printf("INSIDE MATRIX!!\n");
  else
    printf("OUTSIDE MATRIX!!\n");
  return 0;
}
```

About S21sec

S21sec, a company specialising in digital security services and a leader in the sector, was founded in San Sebastian in 2000. It has offices in Madrid, Barcelona and Pamplona and employs 130 information systems security experts. More information about S21sec: http://www.s21sec.com