Anti-debugging Framework Based on Hardware Virtualization Technology

Tengfei Yi, Aijun Zong, Miao Yu, Zhong Ren, Qian Lin, Zhengwei Qi
School of Software, Shanghai Jiao Tong University
{yitengfei,zongaijun,superymk,renzhong,linqian,qizhwei}@sjtu.edu.cn

Abstract

Anti-debug technique is a common security protection mechanism, which is widely used in commercial applications as the protector for executable files. Although there're many such techniques at hand, they all have a common disadvantage that the anti-debug protection cannot guarantee its function. Because the code is running on Ring level 0 or above, the virus application can still manipulate according to the specific anti-debug technique, thus achieving its purpose of obstructing the anti-debugging process. In this paper, a hardware virtualization technology based highly reliable anti-debug framework called VMM (Virtual Machine Monitor) is introduced. Since it runs below the Ring level 0, theoretically every code which runs on this level can be monitored. Experiment shows that under its protection, major Windows debuggers like VC and WinDBG fail to debug the target application, so our purpose is initially achieved.

Index Terms—hardware virtualization; security; anti-debug; VMM

1. Introduction

Nowadays, Windows systems are extensive used around us. However, they can not provide sufficient security and software protection because of the architecture and hardware limitation. Consequently, nearly all the commercial software needs to implement its own additional protection module, raising the total cost on software development.

Under this background, anti-debugging techniques, as a strategy of protecting target application, have been being taken in-depth research, generally based on memory data difference, system difference, CPU difference, etc.

However, the risk of attacking the target software still exists for the following reasons:

First, the hardware architecture is defective, because it make the code running in the privileged mode can access to the whole system, while the code in the user mode can only own a little of resource [1]. That means once the malicious code or analyzing tools are running in the privileged mode, no higher-mode code can be used to stop it.

Second, there are more or less bugs and debugging functions in Windows systems [2], which provide APIs to see other processes' address space as well. So, there is hardly any way to stop commercial software to be cracked.

Fortunately, many new approaches to anti-debugging have appeared with the growing popularity of hardware virtualization [3,4,5,6]. However, these approaches are difficult to be adopted because of their low performance and portability.

Unlike the priors, our design desires no code modification of Windows systems or additional hardware. Indeed, we make use of Intel VT to create a transparent environment for Windows while keeping the ability of monitoring the target application's execution and state. That is a highly reliable anti-debug framework called VMM (Virtual Machine Monitor) based on hardware virtualization technology.

Our work represents the following contributions:

First, by employing the hardware virtualization technology, the in1 and int3 interrupts in the protected application are specially treated, so that the debugger cannot get the status information about the target application.

Second, By employing the hardware virtualization technology, the CR3 block, PCB block and a portion of memory address field of the target application are protected, so that the debugger can not get the information about the target application's memory address.

The other parts of this paper are organized as follows. Part 2 introduces the hardware virtualization technology and then the system architecture. Part 3

introduces the hardware and software debugging principle as well as anti-debug countermeasure. Part 4 talks about how to protect the CR3 block and PCB block in the target application. Part 5 is the experimentation on the theory formulated in Part 3 and Part 4, and makes the conclusion.

2. Hardware virtualization and VMM

Nowadays virtualization technology is getting more and more attention, Enterprise virtualization software, such as VMware, Citrix and Parallels are warmly welcomed by various server vendors, with its applications thriving. However, besides the software component, virtualization technology also requires the hardware support from the bottom layer, since hardware has played a fundamental role in this technology. Chip vendors like AMD and Intel have enhanced the support for virtualization in their next-generation processors.

Virtualization technology is divided into software virtualization technology and hardware virtualization technology. Compared with software virtualization technology, the hardware virtualization technology has many advantages, which break many of the limits set by the pure software virtualization solution.

Hardware virtualization CPU has integrated the specially optimized instruction sets to control the virtualization process. Through these instruction sets, Virtual Machine Monitor (VMM) can improve its performance very easily, thus achieving greater enhancement in performance compared with the software virtualization implementation. Because the virtualization hardware offers a brand new architecture, which supports the direct running of the operating system on it, there's no more need for the binary conversion. Thus the related cost is reduced and the VMM design gets greatly simplified. Furthermore, the VMM can understand the normal code, so its performance becomes more powerful [7].

Let's take a detailed look into the hardware virtualization technology. Because of the support of CPU, VMM can run on higher level of permission than the operating system. The system architecture is illustrated in Figure 1:

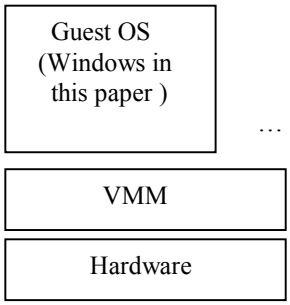

Figure 1. System architecture

To support the hardware virtualization technology, the CPU has integrated a series of virtual machine instructions, through which the operations like opening and closing the virtual machine is done very conveniently.

In the time of running of the virtual machine, the VMM wants to listen to certain events. In order to save data on the spot, both the VMM and Guest OS need a data structure for storage called VMCS control block. The VMCS is an abstraction of Guest virtual machine.

Hardware virtualization is an abstract layer, which separates the physical hardware from the operating system, in a way that the virtual machine is insulated from the host PC and other virtual machines, thus offering higher resource efficiency as well as flexibility.

The VMM we have designed includes the initialization module, virtualization CPU module and VMMU module. Among them, the virtualization CPU module is the core. It is in charge of the control register for processor and state register for some special registers. When Guest OS is running on the virtual machine, it prepares for the necessary data structure and code for creating and maintaining the complete system status information, so as to trap VMM mode from Guest status, or return to a certain Guest OS at any time. It is noted that all the Guest events under the control of VM Exit event processing module, no matter the root cause be accessing critical resource or be exception or interrupt, the function is to find the cause of generating the VM Exit and turn to the security monitor module of the Guest operating system for further processing. This gives our paper sufficient theoretical and experimental basis.

In a word, the VMM proposed in this paper has the obvious characteristic of a layered structure, which is also revised from BluePill project, where the anti-detection part of VMM is deleted and is transplanted to x86 platform to expand its application. Something worth noticing is that the VMM is loaded onto the Windows for function in a way of a driver, which is so-called VMM driver. Although a typical VMM can

support multiple virtual machines, the current design only supports one. No matter how many virtual machines to support, the VMM offers a new approach about software protection. It introduces an additional protection layer but without any change to the existing hardware and operating system.

3. Anti-debugging of Software and hardware breakpoint

3.1. Software breakpoint and anti-debugging

X86 serious CPUs has been providing a special instruction, INT 3, to support debugging from 8086. In other words, this instruction can make CPU break to debugger, helping debugger uses analyze the breakpoints. When debugging, we can insert INT 3 into anywhere a problem would appear, which can make CPU suspends. So, INT 3 is also called breakpoint instruction.

When we use a debugger such as VC6 and WinDBG to set a breakpoint to a certain line of code, the debugger will conserve the first byte of the instruction, and use INT 3 instead(the machine code of INT 3 takes only one byte).

When running to the INT 3 instruction, CPU will jump to the exception handler to make the current application debugged. After that, the exception handler will resolve the breakpoint.

So, the mechanism of software debugging indicates that the INT 3 instruction be shielded, and that we can set the VMM to make the anti-debug come true [8].

3.2. Hardware breakpoint and anti-debugging

IA-32 CPU define 8 debug registers DR0-DR7, and among them, DR0-DR3 are used to be designated linear addresses of the breakpoints. When a application implement to a linear address contained in DR0-DR3, the INT 1 instruction will run, which is similar to the mechanism of software debugging [8].

4. Target Process Protection

In fact, in the face of anti-debug technology, we hope not only disable the breakpoints, but also protect the target application from being accessed by other processes, including the debuggers. We will not distinguish the concept of process and application in this paper, since a process is created when the application is loaded by the OS.

Two important structures or registers need to be considered: PCB and CR3 register.

PCB (Process Control Block) is one of the key structures which hold information of a process on Windows system. Thus OS can use this information to manipulate process. PCB includes the following contents:

(1)Process ID (Internal, External)

(2)Various Register

(3)Process Schedule Information (Process State, Priority, Events)

(4)Process Control Information (Process Data Address, Resource Manifest, Process Synchronizing, Process Communication, Link Pointer)

The CR3 register stores the physical address points to the current process' page directory. When process switches, OS reads the active process' page directory pointer into the CR3 register from its PCB [6], as shown in Figure 2.

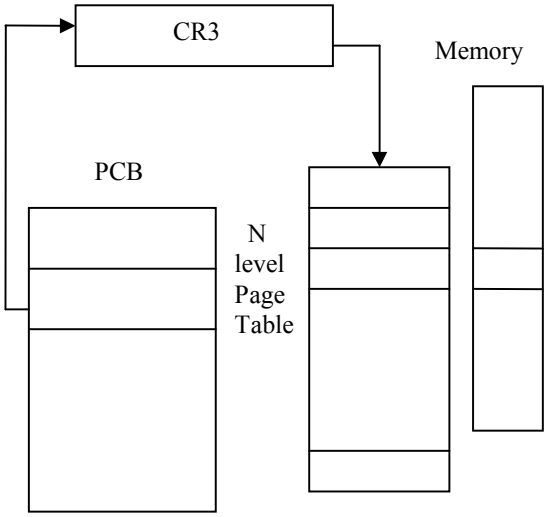

Figure 2. The role of CR3 and PCB

So, the best way to protect one process from being accessed by the other, is to prohibit the target process' PCB from being read, which was difficult to achieve before. However, the level VMM runs below Ring level 0 can make the Process protection come true.

When process A accesses process B, the former requires the CR3 information stored in PCB of the latter. The VMM should be configured to inspect whether the PID of current visited process is identical to that of process B. Only if they have the same value, the access request will be permitted; otherwise, rejected.

5. EXPERIMENTS AND RESULTS

All experiments were conducted on a desktop computer with a 1.83GHz Intel Core2 Duo processor

and 2GB RAM. Windows XP SP3 is selected as the guest operating system.

5.1 Software breakpoint debugging

After entering the following codes in VC, we set a breakpoint at the 3rd line.

```
Main()
   {int i=1,j=2;
   k=i+j;
   …
   }
```

The value of variable i and j were changed at single step in which there was no VMM driver. However, Variable k was not assigned in this whole work, which means the breakpoint worked.

On the other hand, the value of k was assigned to 3 when the VMM driver was settled. From this, the breakpoint did not work and the VMM driver made an effect of anti-debugging.

The principle of hardware breakpoint debugging is similar to the software.

5.2 process protecting

The VC was used to debug the applications as follows:

(1)Started the calculator application and input a string of numbers (e.g. 1234) for observing below.

(2)Started WinDBG and attached it to the calculator application to debug. (chose File>Attach to a process…)

(3)Typed "x calc! g*" in the WinDBG command line to list all the symbols starting with "g" in the calculator application. Pay attention to the line which contains gpszNum.

```
…
01028db0 calc! gpszNum = <no type information>
…
```

(4)Typed "dd calc! gpszNum l1" in the command line and checked the contents of the following symbolic address.

```
01028db0 000a6c00
```

(5)Continued checking the contents of 000a6c00. (input command like "dd 000a6c88")

…

(6)Installed the VMM driver and repeat the step (1)-(5). When it ran to step 3, the output displayed 000000.

It should be noted that the reason why it displayed 000000 is several configures were set in VMM to output 000000 when other applications accessed the calculator application.

In conclusion, the designed framework in this paper make breakpoint disabled assessment from one application to another, including by using the debuggers. We got the anticipative result of protecting target application. In the same way, the anti-debugging tech based on the hardware virtual machine that protects the applications from viruses invading, is proved more reliable than previous anti-debugging tech.

6. References

[1] Russinovich M.E., Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server 2003, Windows XP, and Windows 2000, Microsoft Press, 2005.

[2] Nick L. Petroni and PMichael Hicks. "Automated detection of persistent kernel control-flow attacks", Proceedings of the 14th Computer and communications security, Oct. 2007, Alexandria, Virginia, USA.

[3] Igor Burdonov; Alexander Kosachev and Pavel Iakovenko, "Virtualizationbased separation of privilege: working with sensitive data in untrusted environment", Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems, Mar. 2009.

[4] Artem Dinaburg, Paul Royal, Monirul Sharif, Wenke Lee. Ether, "Malware analysis via hardware virtualization extensions", Proceedings of the 15th ACM conference on Computer and communications security, Oct. 2008.

[5] Payne, B.D., Carbone, M., Sharif, M., Wenke Lee and Lares, "An Architecture for Secure Active Monitoring Using Virtualization Security and Privacy", 2008. SP 2008, *IEEE Symposium*, 18 22 May 2008, Page(s):233 - 247.

[6] Intel. *Intel Trusted Execution Technology Architecture Overview*. September 2006.

[7] Gideon Gerzon, Intel Mobile Group, *Intel VT Processor Virtualization Extensions and Intel Trusted execution Technology*, 2007.

[8] Yinkui Zhang, *Software Debugging*, Electronic Industry Publishing House, Beijing, 2008.