A Hypervisor IPS based on Hardware assisted Virtualization Technology

Junichi Murakami (murakami@fourteenforty.jp)
Fourteenforty Research Institute, Inc.

1. Introduction

Recently malware has become more stealthy and thus harder to detect than ever before. Current malware uses many stealth techniques, such as dynamic code injection, rootkit technology and much more. Moreover, we have seen full kernel mode malware like Trojan.Srizbi.

Many detection tools were released that specialize in kernel mode malware and especially in the detection of rootkits. However, these tools are a cat and mouse game, because they and the malware are executed on the same privilege level.

This is why we developed an IPS based on a hypervisor, which uses features of hardware virtualization. It is executed on Ring-1 and thus runs with higher privileges than the OS layer.

In this paper, we will describe the stealth mechanisms used by recent malware and how to protect against such malware using a Hypervisor IPS.

2. Subversive techniques in kernel space

Joanna Rutkowska proposed a stealth malware taxonomy in November 2006[1]. This paper categorizes malware into four groups, from Type 0 to Type III, as shown below.

2.1 Type 0

Standalone malware, which never changes any system resources. This type of malware might delete personal files, connect to malicious server, open ports waiting for commands from evil users and show other malicious behaviour. However this type doesn't change any system functions such as enumerating files, registry entries, and connections.

Therefore, a detector can detect these easily, if it has the necessary signatures.

2.2 Type I

Malware changes persistent system resources such as code sections in process memory, kernel and BIOS. These resources in memory are never modified by regular execution. Detector can check the memory's integrity using hash algorithms like MD5 and SHA1. The techniques listed below fall into this category.

System register and MSR (Model Specific Register) modification

IDTR, other system-register and MSR hold address of entry points for kernel space. Malware can control the execution path by changing such register's value.

(Well-known) Function pointers modification

IDT (Interrupt Descriptor Table), SDT (Service Descriptor Table) and SSDT (System Service Descriptor Table) are often used by rootkits to control (hide) information regarding system resources like processes, registry entries, connections, drivers, and services.

IDT contains addresses of interrupt and exception handlers. Therefore rootkits can control these events by changing the relevant address. SDT and SSDT contain address of NativeAPI (NtXXX) functions, it is possible to hook those as well.

Code patching

If the process has sufficient privileges, it can overwrite code section in both user-mode process memory and kernel memory. Malware can control the execution path by overwriting jump codes like “mov eax,0xNNNNNNNN; jmp eax”. Detour[2] is known as a technique that makes use of this approach.

2.3 Type II

Malware changes non-persistent system resources such as data sections in process and kernel memory.
The techniques listed below fall into this category.

DKOM (Direct Kernel Object Manipulation)

This technique was presented by Jamie Butler during BH-US-04[3]. In this method, malware manipulates the process list, tokens and other kernel objects directly. For example, the malware may unlink the process which it wants to hide - such as a backdoor trojan in user land - from the process list. On the other hand, the malware may add special privileges to the process by manipulating process tokens.
However, DKOM's capability is limited, because it depends on the implementation of the process that handles the data.

KOH (Kernel Object Hooking)

This technique was presented by Greg Hoglund in January, 2006. The malware modifies function pointers in kernel space such as the Unload routine used by the DriverObject. Of course, every DDK writer knows about DriverObject and DriverObject->Unload. However, there are many kernel objects which contain function pointers in kernel space. A detector has to know about the presence of all such function pointers in order to detect illegal patching.

2.4 Type III

No hooks exist within the system. The malware abuses Virtualization Technology. The three items below are known as Type III rootkits.

BluePill[4]

BluePill was introduced by Joanna Rutkowska at Black Hat USA 2006. This version of BluePill abuses AMD-v technology. Recently, a new version of BluePill was released which supports both Intel VT and AMD-v technologies.

Vitriol[5]

The first rootkit to abuse Intel VT technology is presented by Dino Dai Zovi, at BlackHat USA 2006. However, this rootkit is closed source.

VMM Rootkit Framework[6]

Shawn Embleton posted this rootkit framework, which abuses Intel VT technology, to rootkit.com in August 2007. The source code contains about 2500 lines of code in a single file. It is an excellent starting point in order to learn how creating a VMM works.

3. Virtualization Technology

3.1 CPU Virtualization

In multi tasking environments, general purpose registers should be reserved for each process, since these registers are unique resources for the process.
Similarly, system-registers and some special registers should be reserved for the VMM and each VM, because these registers are unique resources for the OS. Exceptions, which are raised at runtime, should be delivered back to their source.

3.2 Memory Virtualization

The VMM and each individual VM have to internally take control of a different address space. The VMM has to manage these address spaces. It has to take care of memory access between several VMs.

Shadow paging

Basically speaking, shadow paging is an advanced form of address translation from a VA (Virtual Address) to a PA (Physical Address) depending on the CR3 register and the page table which it points to. Therefore, the VMM has to prepare the page table so it relates guest VA to guest PA and guest PA to host PA. The result of this effort is called the SPT (Shadow Page Table). The guest's memory access is translated as follows.

guest VA -> guest page table -> guest PA (VMM VA) -> SPT -> Real PA

If the processor supports EPT (Extended Page Table), this 2-stage translation is automatically done by the MMU. However, EPT is not implemented yet. The VMM should thus implement this translation in software by intercepting the events below.
- Page fault exception
- MOV to CR3
- INVLPG

Details on SPT are described in the Intel Software Developer's Manual Vol. 3B.

3.3 Device Virtualization

The VMM should virtualize device access in addition to CPU and memory.

- Hardware Interrupts

- I/O Instructions
- MMIO (Memory mapped I/O)
- DMA (Direct Memory Access)

3.4 Intel VT

Intel VT is a function to support virtualization within the processor. It works similarly to exception handlers in software. The VMX mode was newly added as an operation mode of CPUs. It is composed of VMX non-root-operations that aid the VM and VMX root-operations that aid VMM. Software can tell the management structure, which is called VMCS, based on what conditions the control should be turned over to VMM. This way implementing VMM is possible in a more natural fashion when using Intel VT.

4. Viton, Hypervisor IPS

Viton is an IPS, which runs outside of the guest. Currently it is not much more than a PoC and was tested on Windows XP SP2 only. It protects the kernel from being modified by forcing immutability on persistent system resources while observing control and system registers modifications. It is based on Bitvisor.

4.1 Bitvisor

The Bitvisor VMM software is developed by the Secure VM Project centered around Tsukuba Univ. in Japan. It is based on Intel VT, and has the following features.

- Open source, BSD License
- Semi-path through model
- Type I VMM (Hypervisor model, like Xen)
- Full scratched, pure domestic production
- Support for 32/64 bits architecture in VMM
- Support for Multi-core/Multi-processor in VMM and Guest
- Can run Windows XP/Vista as Guests without modification
- Support for PAE in the Guest
- Support for Real-mode emulation

4.2 What Viton protects

Viton protects following resources.

- Instructions
  - Detects and blocks all VMX instructions which are raised in the guest
- Registers
  - Watchdog for the IDTR register
  - Locking the MSR[SYSENTER_EIP]
  - Locking the CR0.WP (Write Protect) Bit
- Memory
  - Protect from modification
    - All code sections (R-X) in ntoskrnl.exe
    - IDT
    - SDT
    - SDT.ST (SSDT)

4.3 How Viton works

Viton clears the WR bit in an SPT entry. If CR0.WP is set, even the kernel cannot modify the page. As a result, IDT and SDT, which are contained in the given page are protected from modification.

5. Limitations

Viton doesn't protect the files stored on disk. Attackers can modify files which are executed during the boot sequence like NTLDR, ntoskrnl.exe. If the system needs complete protection, it is necessary to protect these files using another solution. However memory patching attacks targeting such a process or the driver can be prevented by using Viton. In addition, technologies using the TPM such as Intel TXT are already able to protect files executed by the boot sequence such as NTLDR and the BIOS.

6. Appendix

[1] http://www.invisiblethings.org/papers/malware-taxonomy.pdf

[2] http://research.microdot.com/sn/detours/

[3] http://www.blackhat.com/presentations/win-usa-04/bh-win-04-butler.pdf

[4] http://www.bluepillproject.org

[5] http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Zovi.pdf

[6] http://www.rootkit.com/newsread.php?newsid=758

[7] http://www.securevm.org