Clear Hat Consulting, Inc.
www.clearhatconsulting.com

SMM Rootkits

White Paper

By Shawn Embleton & Sherri Sparks

Contents

Introduction

For the past several years, rootkit detection and defense has been an ongoing, complex game of hide and seek between rootkit and security software developers. The emergence of hardware virtualization technology dramatically changed the game and marked a critical divergence in rootkit evolution. Where prior rootkits had co-existed with the OS and exerted control by making modifications to either static or dynamic OS data structures in memory, virtualization opened the door to a new breed of Operating System independent rootkit that had the potential for complete system control without modifiying a single byte in the OS. Virtualized rootkits, however, were limited to newer Intel or AMD processors supporting hardware virtualization extensions. System Management Mode (SMM), a relatively obscure mode on Intel processors used for low level hardware management, provides a similarly stealthy, OS independent environment for rootkits that is viable on the older non-virtualized subset of existing hardware.

The aspects of SMM that that make it a potentially attractive home for rootkits are the fact that it has an isolated memory space and execution environment which can be made invisible to code executing in other processor modes (e.g. Windows protected mode).

Intel System Management Mode

System Management Mode is one of 4 Intel defined processor modes. It is provided for managing low level hardware events like power and thermal regulation. The aspects of SMM that that make it a potentially attractive home for rootkits are the fact that it has an isolated memory space and execution environment which can be made invisible to code executing in other processor modes (e.g. Windows protected mode).

The System Management Mode memory space, referred to as SMRAM, contains the CPU state at the point of entry into SMM, the SMM handler code, and its associated data. The Intel chipset documentation defines 3 locations for SMRAM: Compatible, High Memory Segment, (HSEG), and Top of Memory Segment (TSEG). The compatible region is the default location for SMRAM and ranges from 0xA0000 – 0xBFFFF. The HSEG and TSEG regions provide a write-back cachable SMM memory space. SMRAM is only accessible from code executing in SMM when the D_LCK bit in the chipset SMRAM control register is set. Isolation is enforced by the Memory Controller Hub (MCH). The MCH acts as a gate keeper through which all main memory transactions must pass. When a non-SMM access is detected to SMRAM, the MCH re-routes that access. When dealing with the compatible region, it is re-routed to the VGA framebuffer. An internal processor register called SMBASE

holds the physical address pointer to the beginning of the SMRAM space.

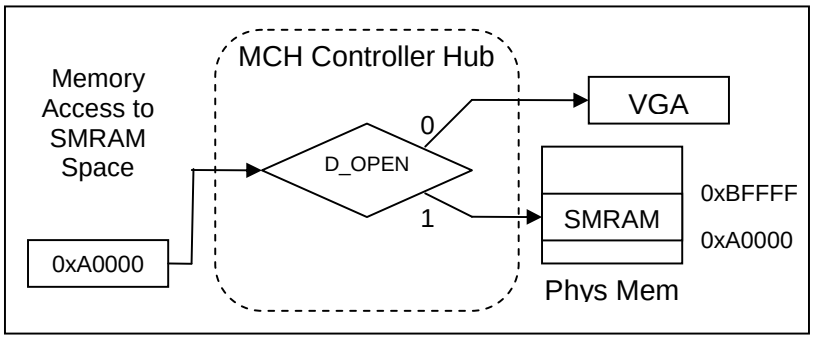

Figure 1: MCH re-routes memory accesses from non SMM code

The CPU enters System Management Mode in response to a System Management Mode Interrupt (SMI). An SMI causes the processor state to be save to the SMRAM space and transfers control to the SMM handler entry point. The System Management Mode handler is typically installed by the BIOS.

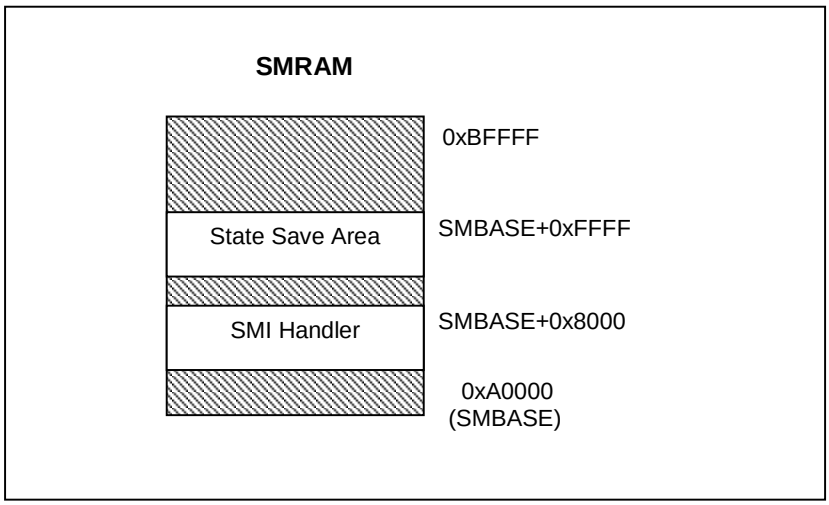

Figure 2: Location and layout of compatible SMRAM region on a 32 bit system

Exploiting System Management Mode

The idea of SMM based malware is not new, but to the best of our knowledge, no one has publically discussed or demonstrated the implementation of an SMM based rootkit. Duflot first discussed using SMM to escalate privilege on OpenBSD systems. He demonstrated an exploit using SMM that allowed an attacker to arbitrarily gain superuser

privileges. This exploit took advantage of the fact that SMM code has unrestricted access to physical memory. Duflot demonstrated that if an attacker can run code in SMM that locates an internal Operating System variable used to determine current privilege level, then he / she can arbitrarily modify it to escalate privileges. Duflot's exploit ,however, was not a rootkit. His goal was privilege escalation, not stealth. The ability to read and write physical memory is only one SMM capability of interest to the SMM rootkit author. A potentially more interesting capability lies in the ability of SMM code to exert unrestricted and unmonitored control over peripheral hardware. BSDaemon, coideloko, and D0nand0n also published an article in Phrack 65 which provides more details on SMM and how to create stable SMM code.

The ability to read and write physical memory is only one SMM capability of interest to the SMM rootkit author. A potentially more interesting capability lies in the ability of SMM code to exert unrestricted and unmonitored control over peripheral hardware.

The rootkit that the authors will be discuss at Black Hat 2008 takes Duflot's work a few steps further by implementing a simple, but functional SMM chipset level keylogger and network backdoor.

Implementing A SMM Rootkit

As mentioned above, Loic Duflot first discussed the general methodology for writing in a new SMM handler. We give the relevant background information here.

Installing A New SMM Handler

The Intel chipset defines 2 registers which are used to configure the behavior of System Management Mode. These are the System Management RAM Control (SMRAM) and the Extended System Management RAM Control (ESRAMC) registers. The SMRAM register is used to control the general visibility and accessibility of the SMM memory space. The layout and location of this register are chipset specific. On most of the systems that we tested, it is located at an offset of 0x9D in the PCI configuration space.

Res.	D_OPEN	D_CLS	D_LCK	GLOBAL SMRAME	0	1	0

Figure 3: Layout of SRAM register on Intel chipsets

The 2 primary fields of interest to the rootkit author are the D_OPEN and D_LCK bits. The D_OPEN bit controls whether or not SMRAM is visible from non-SMM processor modes. If D_OPEN is set, the SMM memory space will be readable and writable outside of SMM (e.g. in Windows protected mode). If it is clear, then the chipset Memory Host Controller (MCH) will

re-route any accesses to this space to the video frame buffer area if they originate from outside SMM mode. In order for a rootkit to install its own SMM handler, it must be able to set the D_OPEN bit. The ability to set D_OPEN (and the other bits in the SMRAM register) depends on the value of the D_LCK bit. Once the D_LCK bit is set, the SMRAM register becomes read-only and D_OPEN is automatically cleared. Furthermore, the only way to clear the D_LCK bit is via a reset.

The SMRAM control register is located in the PCI configuration space defined by the chipset. It may be read and written using port I/O (in / out) instructions over the PCI bus. Reading and writing the SMRAM register is accomplished in 2 steps. In the first step, we must load the PCI configuration address register with a value that identifies the SMRAM register using the *'out'* instruction.

The PCI configuration address register is located on port 0xCF8 and is defined in the chipset documentation as follows:

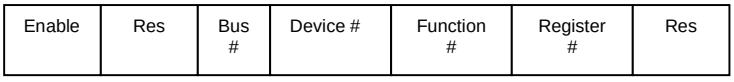

Figure 4: Layout of PCI configuration address register

The second step involves reading the actual data contained in the SMRAM control register. We can do this by reading the PCI configuration data register using the *'in'* instruction after we have set the address register. The data register is located on port 0xCFC.

Installation of an SMM rootkit essentially involves writing in a new SMM handler. The steps in this process can be summarized as follows:

1. On a host machine, an attacker makes SMRAM visible from protected mode for reading and writing by setting the D_OPEN bit in the SMRAM register

2. Once D_OPEN is set, the attacker copies the rootkit SMM handler code to the handler portion of SMRAM as defined by the Intel documentation. By default the SMI handler is located at SMBASE+0x8000.

3. Finally, the attacker clears the D_OPEN bit in the and sets the D_LCK bit in the SMRAM control register. This has the effect of making SMRAM invisible to everything other than the subverted

(rootkit) SMI handler and of locking the SMRAMC register so that it can no longer be modified. As mentioned previously, addressing of the SMRAMC register is chipset specific.

While this process is fairly straightforward in theory, there are several factors that can complicate the implementation. First, it may not be possible to overwrite an existing SMM handler without compromising system stability. During the course of our research, we looked at the SMM handlers on several different machines, including both desktops and laptops. We found that over-writing the SMM handler on desktops was not usually a problem as the desktops did not seem to use SMM. Laptops, however, proved more problematic. Likely, this is because they have more aggressive power management schemes and, consequently, use SMM for legitimate management functions more the desktops. On the laptops that we examined, we generally found it necessary to "hook" the old SMM handler so that we could forward handling to it for non-rootkit related SMI events. The need to preserve the existing handler led to the second problem: that is, the problem of locating space to store the new handler. Because the compatible SMRAM space is limited in size, it may be difficult to find sufficient space to store the rootkit SMM handler. In short, automating this process to work on an arbitrary system is difficult and makes it hard to guarantee stability.

Implementation Of A Chipset Level Keylogger

The implementation of the rootkit handler will depend largely on the intended functionality for the rootkit. As a proof of concept implementation, our rootkit functions as a chipset level keylogger and network backdoor. The implementation of the SMM keylogger can be broken down into three challenges. First, we must be able to intercept the keyboard interrupt from within SMM. That is, user key presses originating from a user in protected mode Windows must be made to invoke the rootkit SMM handler. Second we must be able to sniff the keystrokes from the keyboard's internal buffer. Finally, we should forward the interrupt to the CPU for normal handling.

The first challenge involves intercepting the keyboard interrupt from within SMM and the solution requires an understanding of the Intel Advanced Programmable Interrupt Controller (APIC) architecture.

The Intel Advanced Programmable Interrupt Controller (APIC) is used to manage communication between the CPU, chipset, and external peripheral devices. It consists of two

components: The I/O APIC and the Local APIC (LAPIC). The I/O APIC is located on the motherboard while the Local APIC is integrated into the CPU. There is typically one I/O APIC for each peripheral bus and one Local APIC per CPU. The primary job of the I/O APIC is to route the interrupts it receives from peripheral buses to one or more Local APICs on the system. In turn, each local APIC is responsible for receiving and managing the external interrupts for the CPU that it belongs to. When it receives interrupts, the LAPIC dispatches them to the processor, one at a time, based upon their priorities.

The processor subsequently looks up the handler for the interrupt in the Interrupt Descriptor Table (IDT). Each interrupt is assigned a unique identifier, called a vector. The processor uses this value as an index into the IDT. The Interrupt Descriptor Table is a processor specific data structure containing one entry for each of 255 defined vectors. Kernel rootkits often use IDT *hooking* to intercept processor interrupts and exceptions. This involves replacing the Operating System handler contained in the IDT with a pointer to a malicious *hook* routine. Fortunately, such blatant modifications of the IDT are easily detectable. Detection simply involves enumerating each of the handler pointers and validating that the address is within the range of either the OS kernel or a legitimate system driver. If the address falls outside one of these known ranges, it is flagged as suspicious and a security analyst can conduct further investigations.

The SMM rootkit does not need to make any changes to the IDT in order to intercept interrupts for a peripheral hardware device. Rather than intercepting the interrupt at the processor handling level (IDT), the SMM rootkit can intercept it directly at the chipset level by re-routing the interrupt in the APIC.

Unlike the kernel rootkit described above, the SMM rootkit does not need to make any changes to the IDT in order to intercept interrupts for a peripheral hardware device. Rather than intercepting the interrupt at the processor handling level (IDT), the SMM rootkit can intercept it directly at the chipset level by re-routing the interrupt in the APIC. As mentioned earlier, the main function of the I/O APIC is to receive and route peripheral hardware interrupts to the Local APIC for delivery to the CPU. To accomplish this, the chipset architecture defines an I/O APIC *Redirection Table*. The *Redirection Table* contains a dedicated entry for each interrupt pin. It is used to translate the physical, hardware signal into an APIC message on the APIC bus. The table can be used to specify the interrupt destination, vector, and delivery mode. Our rootkit is primarily interested in the delivery mode field. The *Fixed* delivery mode is commonly used for interrupts. It automatically forwards the interrupt to the LAPIC's of all processors listed in the destination field. In order to intercept the keyboard interrupt, our rootkit changes the delivery mode of the keyboard IRQ from *Fixed* to *SMI*. This causes the I/O APIC to generate an SMI in response to

keyboard interrupts. This SMI causes our rootkit SMM handler to run and gives us the opportunity to sniff the contents of the keyboard buffer and send it out in network packets. The keyboard data is extracted by reading the data directly from the keyboard's internal data register. After extracting the keyboard data in the SMM handler we must deliver the interrupt to the CPU so that the OS can handle it normally. Otherwise, the keyboard will no longer function. We can use the Local APIC's ability to issue interprocessor (IPI) interrupts for this purpose.

The LAPIC documentation defines an Interrupt Command Register (ICR). Using this register it is possible to send an interrupt to one or more processors, including self. As in the I/O APIC's Redirection Table, the destination, vector, and delivery mode are all specifiable. When the lower 4 bytes of the ICR are written to, the LAPIC generates the IPI message and sends it out over the system bus. From within our SMM handler, we re-issue the interrupt with a destination of self and a fixed delivery mode by writing to the ICR. Therefore, the keyboard interrupt is delivered to the processor in the normal manner as soon as we exit from SMM mode. The following figure illustrates how the SMBR intercepts a keystroke signal and forwards it to the CPU.

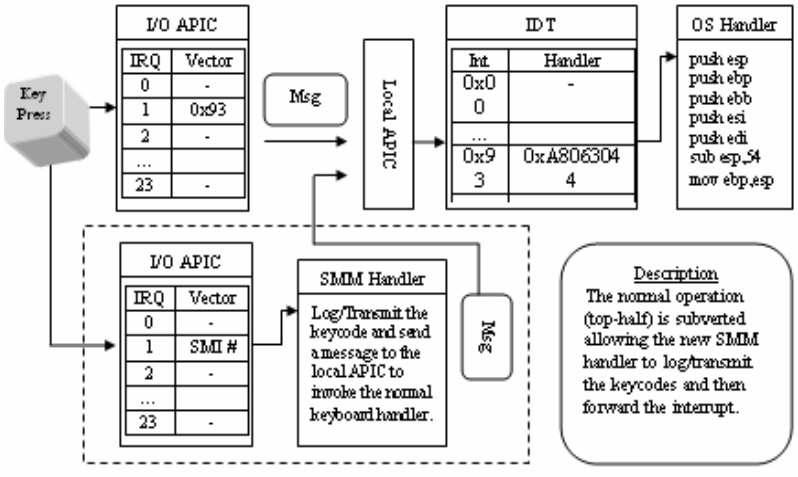

Figure 5: Normal and subverted keyboard interrupt path

Our rootkit intercepts and stores keystrokes in a buffer located in SMRAM. When full, we use the chipset LAN controller to transmit the key data to an external IP address. Using a buffer as opposed to sending the keystrokes immediately allows more variability in when to send the data. For example, a rootkit could use traffic shaping techniques to stealthily blend in with existing network activity. The details of our chipset level backdoor will be discussed in

a followup whitepaper for the other talk that we will present at Black Hat this year “Deeper Door: A chipset level network backdoor”.

VMM vs. SMM Rootkits

VMM and SMM Rootkits are both Operating System independent, stealthy, and hardware specific, yet their vulnerable system spaces are drastically different. Virtualization rootkits can only exist on processors supporting Intel or AMD virtualization extensions. This limits them to relatively new processors (less than 1-2 years old). In contrast, SMM rootkits are more likely to exist on older processors containing older BIOS versions (more than 2 years old). This is due to the fact that many newer BIOS have set the D_LCK bit in the SMRAM control register rendering SMRAM inaccessible outside the BIOS. Nevertheless, SMM provides an alternative to the VMM rootkit with the potential to offer a similar level of stealth and control. There are still a significant number of older systems that don’t support virtualization extensions. On these systems, an SMM rootkit might be viable where a VMM rootkit is not.

SMM provides an alternative to the VMM rootkit with the potential to offer a similar level of stealth and control. There are still a significant number of older systems that don’t support virtualization extensions. On these systems, an SMM rootkit might be viable where a VMM rootkit is not.

The operational environment and focus of control also differs substantially between the two forms of OS independent malware. VMM rootkits operate in a protected mode environment with paging enabled, while SMM rootkits operate in an environment similar to real mode without paging. While both forms of rootkits can effectively control an infected system without modifying their host OS, they do so in different ways. VMM rootkits have the upper hand in terms of flexibility. They can intercept a greater number of higher level processor events like interrupts, memory accesses, debug and control register read / write and privileged instruction execution. SMM rootkits have a great deal of control over peripheral hardware, but their interception is limited to lower level hardware events and they have very little control over processor specific events like memory access and instruction execution. Implementation of both VMM and SMM rootkits is complex, but a stable SMM rootkit may be slightly more so because documentation is not as easy to find or as clearly laid out as the documentation for Intel and AMD virtualization extensions.

Discussion & Defense

To defend against SMM rootkits, we can get some ideas by looking at some of the suggested detections for VMM rootkits.

One proposed detection for the virtualized rootkit is to look at discrepancies in TLB or cache access patterns. This approach will be of limited value against the SMM rootkit. First, TLB access patterns are not relevant because paging is not enabled in the SMM environment. Secondly, the SMRAM memory space is located in non cacheable memory by default (although there are alternative HSEG and TSEG locations defined in the chipset documentation which do allow caching).

Furthermore, one might suggest moving detection off the CPU onto another hardware device that has access to physical memory. The problem with this approach is that the chipset arbitrates all external device communication to and from physical memory through the Memory Controller Hub (MCH). As a result SMRAM remains inaccessible to any devices residing on the system bus.

More promising heuristics for SMM rootkits may include timing based detections and the presence of abnormalities in the I/O APIC redirection table. Timing based detections similar to those proposed for VMM rootkits might be able to be applied to SMM rootkit detection. The time-stamp counter does appear to be updated during an SMI although it is likely possible for a SMM rootkit to cheat in a manner similar to VMM rootkits. It is also important to remember that SMM is used for legitimate reasons on many systems and that the presence of an SMM handler is not, by itself, sufficient to suggest compromise. Finally, the I/O APIC redirection may be checked for interrupts that have been re-routed to SMI. A rerouted interrupt may also be considered a “red flag”, but is also insufficient by itself to suggest compromise. There are legitimate reasons to route an interrupt to an SMI. One such use is to provide legacy keyboard and mouse support for USB devices

SMM rootkits suffer From a number of limitations which make them unlikely to appear outside of specific, targeted attacks where the attacker has some knowledge of their adversary’s hardware.

Duflot’s recommendation that chipset vendors lock the D_LCK bit in the BIOS still remains the best methods to ensure that an SMM rootkit can’t be easily installed. While newer systems appear to have locked this bit, many older systems remain vulnerable.

While SMM rootkit detection may appear to be a bleak prospect, it is redeemed by the fact that SMM rootkits suffer from a number of limitations which make them unlikely to appear outside of specific, targeted attacks where the attacker has some knowledge of his / her adversaries hardware. The SMM rootkit is, by nature, very hardware specific. This hardware specificity includes both the chipset (SMM related register offsets tend to vary by chipset) and any peripherals that the rootkit intends to interact with.

Implementation of this type of rootkit is further complicated by the lack of OS driver and debugging support. Writing chipset level peripheral driver software in a real-mode like environment is a non trivial task. Finally, like the VMM rootkit, SMM rootkit is an in-memory rootkit. That is, it lacks persistence across reboots, however, this limitation could probably be overcome by storing the rootkit code in the BIOS or a boot ROM with the tradeoff of even greater hardware dependence.

About the Authors

Sherri Sparks is President of the Florida company, Clear Hat Consulting, Inc. Currently, her research interests include offensive / defensive stealth code technologies and digital forensics. She has spoken at Black Hat on these topics and has taught the Black Hat Offensive Aspects of Rootkit Technology. Her published articles have appeared in Usenix Login; ACSAC, Security Focus, and Phrack magazine. With an increasing involvement in providing consulting / training services for independent clients, she co-founded the company Clear Hat Consulting, Inc. in early 2007. Clear Hat Consulting specializes in Windows kernel and hypervisor research and development as it related to stealth malware technology, digital forensics, and other custom software security solutions.

Shawn Embleton is the Vice President and CTO of the Florida company, Clear Hat Consulting, Inc. Shawn spoke at Black Hat in 2006 on the topic of using evolutionary computation for automated vulnerability analysis and co-authored a prototype intelligent fuzz testing tool, named Sidewinder. During 2007, Shawn co-taught the Black Hat Offensive Aspects of Rootkit Technology class with Sherri Sparks and co-founded Clear Hat Consulting, Inc. Some of his current interests include hardware virtualization technology and chipset level rootkit technology.

References

1. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide, Part 2. May 2007.
2. Intel Corporation. Intel 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide, Part 1. May 2007.
3. Intel Corporation. Intel 82801DB I/O Controller Hub 4 (ICH4). May 2002.
4. Intel Corporation. Intel 845GE/845PE Chipset Datasheet.

Oct. 2002.
5. 8042 Keyboard Controller.. http://heim.ifi.uio.no/~stanisls/helppc/8042.html
6. L. Duflot, D. Etiemble, and O. Grumelard. Using CPU System Management Mode to Circumvent Operating System Security Functions. In. DCSSI 51 bd. De la Tour Maubourg 75700 Paris Cedex, France. 2007.
7. T. Garfinkel, K. Adams, A. Warfield, and J. Franklin. Compatibility is Not Transparency: VMM Detection Myths and Realities. In HotOS XI: 11th Workshop on Hot Topics in Operating Systems, 2007. USENIX.
8 J. Rutkowska. Subverting Vista Kernel for Fun and Profit. Presented at Black Hat USA, Aug. 2006.
9. J. Rutkowska. IsGameOver() Anyone? Presented at Black Hat, USA. Aug 2007.
10. J. Heasman. Implementing and Detecting an ACPI BIOS Rootkit. Presented at Black Hat, Federal. 2006.
11. BSDaemon, coideloko, and D0nand0n. “Using SMM for ‘Other Purposes’”. In Phrack Volume 0x0C, Issue 0x41, Phile x004.