A New Breed of Malware

The SMM Rootkit

By Shawn Embleton & Sherri Sparks

OS Dependent Rootkits

- Make changes to the host OS
- Hooking
  - Hacker Defender, NTRootkit
- Direct Kernel Object Manipulation (DKOM)
  - FU Rootkit
- Memory Subversion
  - Shadow Walker Rootkit

OS Dependent Rootkits Detection

- Changes to host OS can be detected
  - VICE - detects hooking behavior
  - System Virginity Verifier - in memory integrity checker
  - Rootkit Revealer - cross view detection

OS Independent Rootkits

- Hardware Virtualization Rootkits
  - Bluepill (AMD) – Joanna Rutkowska
  - Vitriol (Intel) – Dino A. Dai Zovi
- BIOS Rootkits
  - Proof of concept ACPI BIOS Rootkit – John Heasman
- SMM Rootkits

OS Independent Rootkits Detection

- OS Independent Rootkits Detection
  - No changes to host OS - Detection Methods are usually indirect
- Hardware Virtualization Rootkits
  - Direct Timing - Read time stamp counter before and after VM Exit event
  - Indirect Timing - Cache / TLB discrepancies
  - Memory Footprint - If not using memory virtualization
- BIOS Rootkits
  - Prevent BIOS reflashing
  - Require signed BIOS

System Management Mode (SMM)

- One of 4 Intel Processor Modes
  - Real, V8086, Protected, SMM
- Useage - Low level Hardware Control
  - Power management
  - Thermal Regulation
  - Legacy Device Support (USB vs. PS2 Keyboard)
- Environment – Similar to "Unreal" Mode
  - Default 16 bit (can access 32 bit flat address space w/ override prefixes)
  - No paging
  - Non preemptible

Why is SMM attractive to rootkits?

- Isolated memory space and execution environment that can be made invisible to code executing in other processor modes (i.e. Windows Protected Mode)
- No concept of “protection”
  - Can access all of physical memory
  - Can execute all instructions, including privileged instructions
- Chipset level control over peripheral hardware
  - Intercept interrupts without changing processor data structures like the IDT
  - Communicate directly with hardware on the PCI bus

Exploiting SMM- Related Work

- Duflot first discussed the idea of SMM based malware
  - Demonstrated using SMM to escalate privilege on OpenBSD systems
  - New SMM handler modified variable that determined current privilege level to escalate privilege
- BSDaemon, coideloko and D0nand0n - phrack 65 article "Using SMM for 'Other Purposes' "
  - More details on SMM workings and thoughts about how to create stable SMM code

SMM Rootkits?

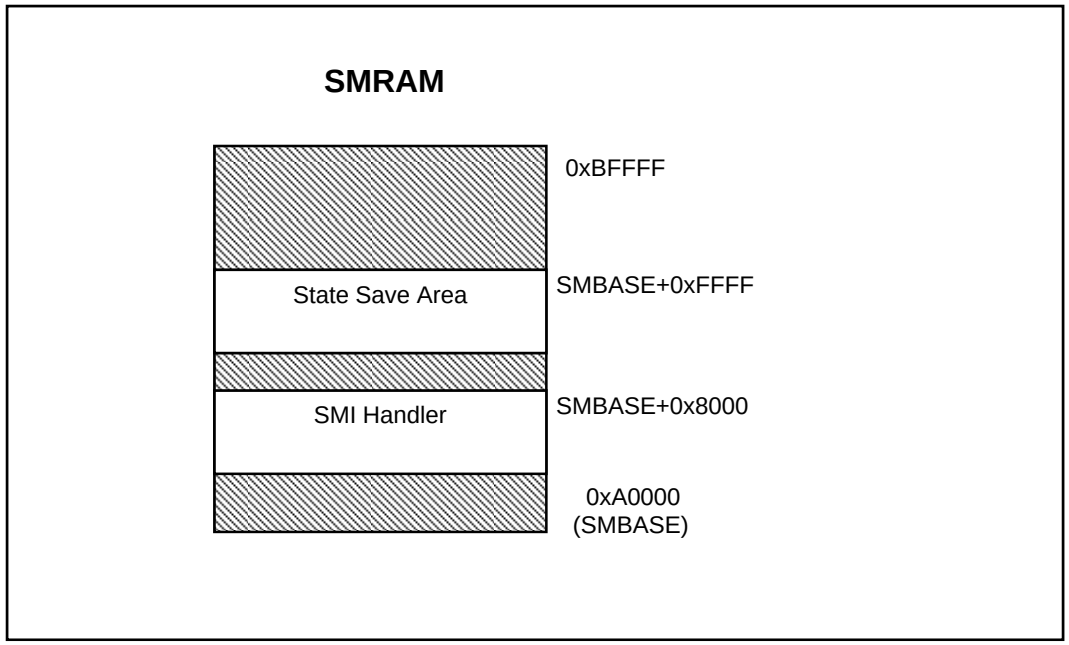
- Previous work was not a “rootkit”
  - One of the more interesting aspects of SMM code is its ability to exert unrestricted & unmonitored control over peripheral hardware
  - We build on previous work to develop an SMM handler with more “rootkit like” functionality.
    - Invisible Keylogging
    - Ability to invisibly send logged packets over network to remote host from inside SMM

SMRAM

- The System Management Mode memory space
  - Contains the CPU state at point of entry into SMM
  - SMM handler Code
  - SMM handler Stack & Data
- Location
  - Compatible (default) 0xA0000-0xBFFFF
  - HSEG or TSEG
    - Provide larger, write-back cacheable SMM memory space
  - An internal processor register called SMBASE holds the physical address of the beginning of the SMRAM space

SMRAM Location & Layout

- Location & Layout of compatible SMRAM region on 32 bit systems



SMM State Save Map

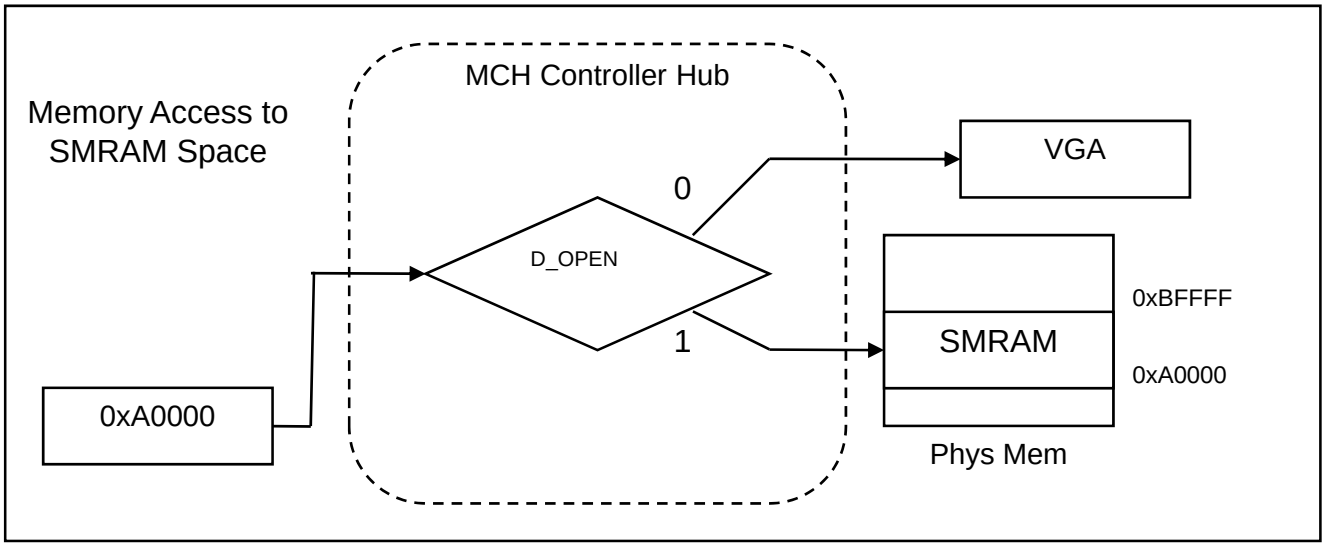
Table 24-1. SMRAM State Save Map

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FFCH	CR0	No
7FF8H	CR3	No
7FF4H	EFLAGS	Yes
7FF0H	EIP	Yes
7FECH	EDI	Yes
7FE8H	ESI	Yes
7FE4H	EBP	Yes
7FE0H	ESP	Yes

Offset (Added to SMBASE + 8000H)	Register	Writable?
7FDCH	EBX	Yes
7FD8H	EDX	Yes
7FD4H	ECX	Yes
7FD0H	EAX	Yes
7FCCH	DR6	No
7FC8H	DR7	No
7FC4H	TR[1]	No
7FC0H	Reserved	No
7FBCH	GS[1]	No
7FB8H	FS[1]	No
7FB4H	DS[1]	No
7FB0H	SS[1]	No
7FACH	CS[1]	No
7FA8H	ES[1]	No
7FA4H	I/O State Field, see Section 24.7	No
7FA0H	I/O Memory Address Field, see Section 24.7	No
7F9FH-7F03H	Reserved	No
7F02H	Auto HALT Restart Field (Word)	Yes
7F00H	I/O Instruction Restart Field (Word)	Yes
7EFCH	SMM Revision Identifier Field (Doubleword)	No
7EF8H	SMBASE Field (Doubleword)	Yes
7EF7H - 7E00H	Reserved	No

SMRAM Isolation

- SMRAM isolation is enforced by the MCH
- Non SMM accesses are re-routed to VGA frame buffer region when D_LCK bit in the chipset SRAMC control register is set



Entering / Exiting SMM

- SMM is entered in response to an System Management Mode Interrupt (SMI)
- On an SMI
  - Processor state is saved to SMRAM space
  - Control is transferred to SMM handler entry point (SMBASE + 0x8000)
  - Handler returns control by executing a RSM instruction

Implementation

- Rootkit Installation
- A Chipset Level Keylogger
- A Chipset Level Network Backdoor

Rootkit Installation

- Overview – PCI Configuration Space
- Unlocking / Locking SMM
- Opening SMRAM for Writing
- Writing in a new SMM handler

PCI Configuration Space

- 2 Registers used to access chipset registers in the PCI configuration Space
- PCI Configuration Address Register
  - Located on port 0xCF8

Enable	Res.	Bus #	Device #	Function #	Register #	Res.

- PCI Configuration Data Register
  - Located on port 0xCFC

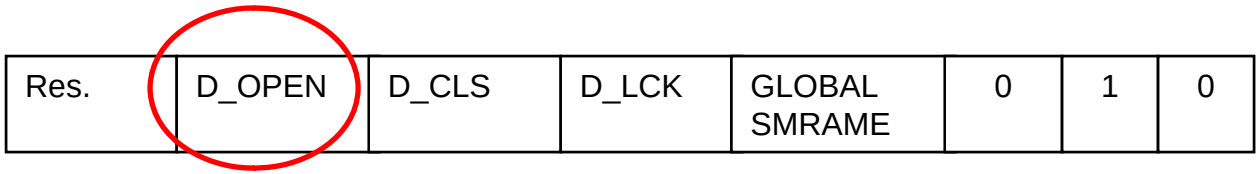
SRAMC

- Used to control visibility and accessibility of SMRAM space
- Layout and location are chipset specific
  - Seems to be located at offset 0x9D in the PCI configuration space on a lot of systems (but not all)
- SRAMC register is read and written via the PCI configuration space address & data registers

Res.	D_OPEN	D_CLS	D_LCK	GLOBAL SMRAME	0	1	0

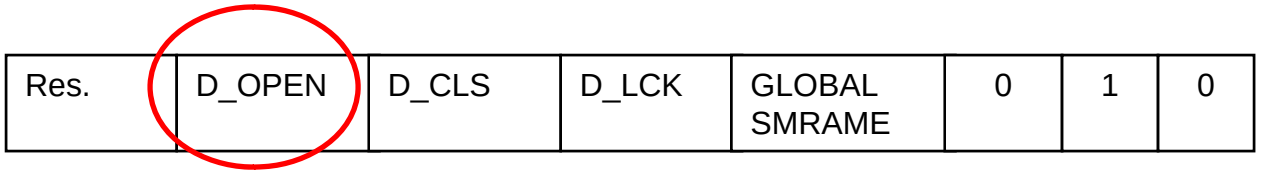
Opening SMRAM for writing

- D_OPEN controls weather or not SMRAM is visible from non SMM processor modes
- If D_OPEN is set, SMM is readable / writable outside SMM
- If D_OPEN is clear, MCH reroutes accesses to frame buffer
- Rootkit must set the D_OPEN bit to install new handler



Locking & Unlocking SMRAM

- Ability to set D_OPEN (and other bits in SMRAM register) depends on D_LCK
- If D_LCK is set, the SMRAM control register becomes read-only and D_OPEN is automatically cleared
- Only documented way to clear D_LCK is via a reset



Rootkit Installation Summary

- Installing a SMM rootkit involves replacing or “hooking” the existing SMM handler
  1. Make SMRAM visible from protected mode by setting the D_OPEN bit in the SMRAM register
  2. Copy rootkit handler code into the handler portion of SMRAM (default is SMBASE + 0x8000)
  3. Clear the D_OPEN bit and set D_LCK so that detection software can’t scan SMM and find the rootkit code

Caveats

- Simple in theory, but some complicating factors…
  - Overwriting the handler may cause system instability (especially on laptops)
  - Need to “hook” original handler
    - May be difficult to find space to store the new handler in an automated way and guarantee stability

Chipset Level Keylogger

- 3 Challeneges
  1. We must be able to intercept the keyboard interrupt from within SMM
  2. We must be able to sniff keystrokes from keyboard's internal buffer
  3. We must be able to forward the keyboard interrupt to the CPU so the keyboard continues to work normally

Intercepting the Keyboard Interrupt

- Need to understand APIC architecture
- 2 Components
  - I/O APIC
  - Local APIC

I/O APIC

- I/O APIC
  - located on motherboard
  - 1 I/O APIC for each peripheral bus
  - Primary job is to route interrupts from peripheral buses to one ore more Local APICs

Local APIC

- Local APIC (LAPIC)
  - Integrated into CPU
  - 1 LAPIC per CPU
  - Receives and manages external interrupts from the I/O APIC and sends them to the processor
  - Processor looks up handler for interrupt in the IDT and calls it

Intercepting the Keyboard Interrupt

- Normal rootkits intercept interrupts by replacing the handler in the IDT
- Looking for abnormal pointers (e.g. pointing outside the OS) is one heuristic method of rootkit detection
- SMM rootkits can't (and do not need) to make any changes to the IDT
- We can intercept interrupts much earlier at the I/O APIC level…

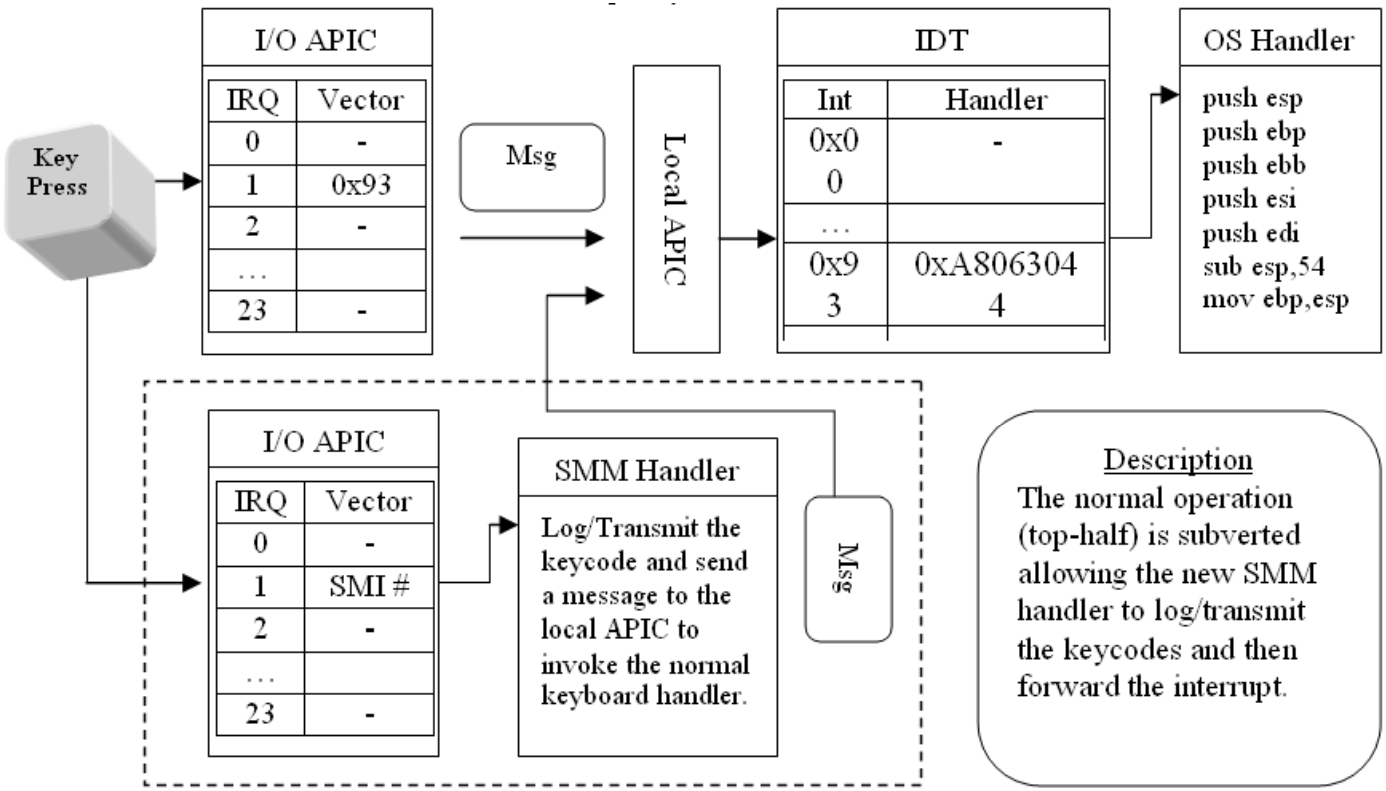
The Redirection Table

- I/O APIC defines Redirection Table for routing peripheral hardware interrupts
- Contains a dedicated entry for each interrupt pin
- Can be used to specify destination, vector, and delivery mode
- Most interrupts use the Fixed delivery mode
- We change the delivery mode for the keyboard IRQ from fixed to SMI
  - Causes the keyboard to generate SMI in response to keyboard interrupts which we can catch in our rootkit SMM handler
  - Keystrokes are then extracted from the keyboard data buffer

Forwarding the keystrokes

- After extracting the keyboard data, we need to deliver the interrupt to the CPU so the OS can handle it normally
- Use Local APIC's ability to issue interprocessor interrupts
  - LAPIC defines an Interrupt Command Register (ICR)
  - When the lower 4 bytes of the ICR are written the LAPIC generates an IPI and sends it out over the system bus
  - We can re-issue the keyboard interrupt with a destination of self and fixed delivery mode by writing to ICR

Interrupt Redirection

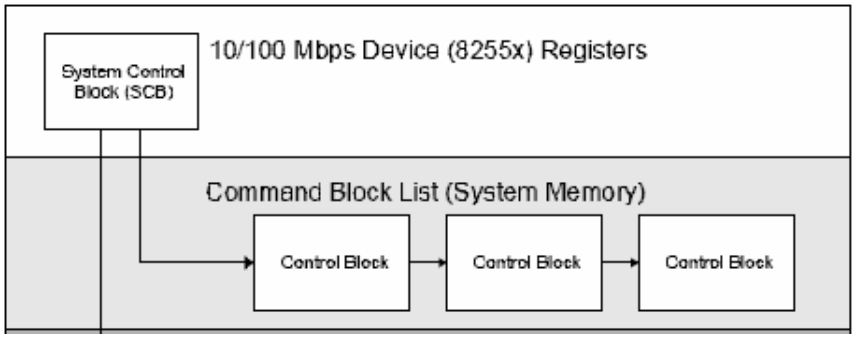


Network Backdoor

- Surprisingly easy… We just need to write to a few registers on the network card (also located in the PCI configuration space)
- Developed for Intel 8255X Chipset
  - Tested on Intel Pro 100B and Intel Pro 100S cards
  - Lots of other cards compatible with the 8255X chipset
  - Open documentation for Intel 8255X chipset
- See our “Deeper Door” talk / slides for details

Data Exfiltration – Sending data out

1. Build the data packet
2. Build A Transmit Command Block (TCB)
3. Check that the LAN Controller is idle
4. Transmit Command Block into System Control Block
5. Write CU_start into the System Control Block to initiate packet transmission



VMM vs. SMM Rootkits

- Virtualization Rootkits
  - Need Intel or AMD hardware support (>2006 systems)
  - Operating Environment - 32 bit protected mode w/ paging
  - OS Independent – YES
  - Able to hide memory footprint - YES, with memory virtualization
    - Control - CPU (processor interrupts, debug and control register accesses, privileged instructions)
    - SMM Rootkits
- SMM Rootkits
  - Must have SMRAM unlocked (< 2005 systems appear vulnerable)
    - Operating Environment - 16 bit unreal mode w/o paging
    - OS Independent - YES
    - Able to hide memory footprint - YES, by default
    - Control - Chipset (IRQ's including keyboard, mouse, NIC, USB, Disk)

Limitations

- D_LCK
- Chipset and hardware dependent implementation
  - Best suited to a highly advanced, targeted attack where the attacker has some knowledge of his / her adversary's hardware
- Handler has to be written in ASM
  - No OS driver or debugging support
  - Unable to survive reboot

Defense (1)

- What about TLB / Cache discrepancies?
  - Proposed detection for VMM rootkits
  - Not as useful for SMM rootkits
    - Paging is not enabled
    - Default SMM memory space is non cachable
- What about moving detection off the CPU to another device with access to physical memory?
  - MCH arbitrates all external device communication to and from physical memory
  - SMM (when locked) remains inaccessible to any devices residing on the system bus

Defense (2)

- What about timing abnormalities?
  - Maybe useful – The time stamp counter does appear to be updated during a SMI
  - Likely is possible for a SMM rootkit to cheat in a similar manner to VMM Rootkits
- What about changes to the I/O Redirection Table
  - Good idea to check chipset structures like the Redirection table for suspicious changes
  - Still, a keyboard interrupt that has been routed to SMI is probably insufficient to suggest compromise
    - Keyboard Interrupt to SMI routing is used legitimately to provide legacy keyboard and mouse support for USB devices

Defense (3)

- Duflot's recommendation that D_LCK be set in the BIOS still remains the best method to ensure an SMM rootkit can't be easily installed
  - Many newer systems have locked this bit
  - Many older systems remain unlocked. A lot of these systems don't support Virtualization. On these systems, an SMM rootkit might be a viable option.
  - Of course, we can always reflash the BIOS or the boot rom for an installed peripheral PCI card, but this is a lot more complex and makes the implementation even more hardware dependent

DEMO