The Memory Sinkhole

Christopher Domas
xoreaxeaxeax@gmail.com

July 20, 2015

Abstract— **In x86, beyond ring 0 lie the more privileged realms of execution, where code is invisible to AV, we have unfettered access to hardware, and can trivially preempt and modify the OS. The architecture has heaped layers upon layers of protections on these 'negative' rings, but 40 years of x86 evolution have left a labyrinth of forgotten backdoors into the ultra-privileged modes. Lost in this byzantine maze of decades-old architecture improvements and patches, there lies a design flaw that's gone unnoticed for 20 years. Exploiting the vast, unexplored wasteland of forgotten x86 features, we demonstrate how to jump malicious code from ring 0 into the deepest, darkest realms of the processor. The attack is performed with an architectural 0-day built into the silicon itself, and directed against a uniquely vulnerable string of code widely deployed on modern systems.**

I. Introduction

THE x86 architecture is traditionally divided into "rings" of privilege, with ring 3 designated the least privileged realm of execution, and ring 0 the most. As the architecture evolved, and deeper levels of privilege became necessary, additional privilege separation mechanisms were developed to confine and restrict ring 0 code from even more powerful modes of execution, colloquially dubbed the negative rings. Ring -1, more commonly known as the hypervisor, is capable of preempting and isolating ring 0 code. Ring -2, System Management Mode (SMM), can further preempt ring -1, has unrestricted access to platform hardware, and in many cases can bypass Trusted Execution Technology (TXT), positioning it as the most privileged level of execution on modern x86 processors. Due to an extreme potential for abuse, SMM is protected through innumerable security mechanisms. However, the complexity of the architecture precludes the simple separations found in higher rings, and SMM security circumventions can be constructed through elaborate configurations of unexpected architectural features.

II. SMM Security Overview

The System Management Mode security model is based upon the premise of a secure and protected region of memory, the System Management RAM (SMRAM). In concept, SMM code resides exclusively within SMRAM, and SMRAM is accessible only while the processor is in SMM. In this setup, SMM code can access everything else on the system, but nothing else on the system can access SMM code. The Memory Controller Hub (MCH), sitting between the processor core and memory, enforces the SMRAM separation. If the processor is not in SMM, the MCH blocks access to SMRAM. If the processor is in SMM, the MCH allows it.

III. Security Violation

The x86 Local Advanced Programmable Interrupt Controller (LAPIC, hereafter referred to simply as the APIC) is tasked with managing interrupt events sent to the processor. Originally a separate circuit, the APIC was integrated with the processor silicon in the P5 microarchitecture. To allow rapid access and flexibility in managing the APIC, the chip's registers were mapped into the processor's memory at the 4KB region between 0xFEE00000 and 0xFEE01000. This inadvertently caused conflicts with software already using this memory range for other purposes. To resolve this issue, the P6 family of processors extended the APIC, to allow remapping the registers to another region of memory. This capability corrected a rare issue in legacy systems, and is neither used nor needed by modern processors; however, modern processors continue to support the remappable APIC feature.

The ancient ability to relocate the APIC registers introduces a complex vulnerability in an entirely unrelated component of the processor architecture – System Management Mode. If the APIC register window is moved to overlap the SMRAM range, memory accesses that should be sent to the MCH for adjudication are instead prematurely accepted by the APIC, and never received by the MCH. This provides ring 0 code a small, indirect influence over SMM, and violates the fundamental architectural separation of the two execution modes.

IV. The Memory Sinkhole

The APIC register window is fixed to a 4KB range, and required to be aligned on a 4KB boundary. The processor has some limited control over the APIC registers, but the vast majority of the 4KB register window is hardwired to 0. In practice, this offers ring 0 code the ability to "sinkhole" a single page of SMRAM by relocating the APIC – memory reads from the region return 0, and memory writes are discarded. The course granularity of the APIC position, combined with the inability to effectively control the APIC data, make the vulnerability extremely difficult, but not impossible, to apply in practice.

V. The SMM Handler

In order to escalate execution from ring 0 to the far more powerful ring -2, it is useful to first examine SMM code for attack vectors. SMM code is installed during the boot process by system firmware, the diversity of which typically precludes a widespread attack. However, select components of system firmware are derived from a set of Unified Extensible Firmware Interface (UEFI) template code provided by Intel. Such is the case for the initial SMM entry point, which is almost universally deployed on modern systems. An attack directed against this specific code sequence achieves the widest possible coverage. The template SMM entry point is provided in Figure 1.

```
8000 mov  bx, off:unk_8091  ; load offset to GDT descriptor
8003 mov  eax, cs:0FB30h    ; load physical address of GDT
8008 mov  edx, eax
800B mov  ebp, eax
800E add  edx, 50h ; 'P'
8012 mov  [eax+42h], dx     ; initialize segments in GDT
8016 shr  edx, 10h
801A mov  [eax+44h], dl
801E mov  [eax+47h], dh
8022 mov  ax, cs:0FB38h     ; load expected size of the GDT
8026 dec  ax                ; decrement total size
8027 mov  cs:[bx], ax       ; save size to the GDT descriptor
802A mov  eax, cs:0FB30h    ; reload GDT base address
802F mov  cs:[bx+2], eax    ; save base address to descriptor
8034 db   66h
8034 lgdt fword ptr cs:[bx] ; load new GDT
8039 mov  eax, 0
803F mov  cr3, eax
8042 mov  eax, 668h
8048 mov  cr4, eax
804B mov  ax, cs:0FB0Eh     ; load expected lmode cs selector
804F mov  cs:[bx+48h], ax   ; patch selector into lmode jmpf
8053 mov  ax, 10h           ; load hardcoded pmode cs selector
8056 mov  cs:[bx-2], ax;    ; patch selector into pmode jmpf
805A mov  edi, cs:0FEF8h    ; load smbase
8060 lea  eax, [edi+80DBh]  ; compute offset of insn at 80db
8068 mov  cs:[bx+44h], eax  ; patch offset into lmode jmpf
806D lea  eax, [edi+8097h]  ; compute offset of insn at 8097
8075 mov  cs:[bx-6], eax    ; patch offset into pmode jmpf
807A mov  ecx, 0C0000080h
8080 mov  ebx, 100011b
8086 mov  cr0, ebx          ; switch to 16 bit pmode
8089 jmp  large far ptr 0:0 ; switch to 32 bit pmode
```

Fig. 1. The template implementation of the SMM entry point, widely deployed on modern systems. Commented for clarity.

There are 2 to 3 common variations to the SMM template code shown above – it appears the SMM entry point has been updated once or twice in the past decade, and the attack needs to be adapted to other versions. The SMM handler entry above appeared to be the most common in our research.

VI. Privilege Escalation

The template SMM handler is verifiably correct and secure from any normal attack vector, but is, by pure coincidence, uniquely vulnerable to the memory sinkhole attack. Critically, the handler loads key data from a structure located at SMM address 0xFB00. It does this at instructions 8003, 8022, 802A, 804B, and 805A above. As this structure resides inside SMRAM, ring 0 cannot access or modify it through any normal means. However, prior to the SMI, if the local APIC is remapped to overlap this structure, these instructions will fetch registers from the APIC, rather than data from memory. These registers in the APIC are hardwired to 0, and cannot be changed, meaning the reads at 8003, 8022, 802A, 804B, and 805A can be configured, from outside of SMM, to read only 0's. This gives a very limited control over SMM execution, but designed correctly, allows forcing a malformed GDT and far jump generation in the SMM code, and causes execution to jump outside of SMRAM, permitting malicious ring 0 code to hijack SMM.

With the APIC configured to overlap the SMM structure at 0xFB00, execution proceeds as follows: the read at 8003 loads 0 as the base address of the GDT; the subsequent instructions set up GDT descriptors in a non-existent GDT near physical address 0; the instruction at 8022 loads the size of the GDT from memory, and the APIC overlap causes the size to be incorrectly read as 0; the dec ax instruction at 8026 recomputes the size of the GDT as 0xFFFF, and 8027 saves the size to a GDT descriptor; 802A loads 0 as the base address of the GDT, and saves it to the same descriptor; the lgdt instruction at 8034 loads the malformed descriptor, placing the new GDT outside of SMRAM, and under the attacker's control. With this, malicious ring 0 code can control the memory layout when new segment selectors are loaded in SMM, which will occur on the far jump at 8089. After the self-modifying code at 8075 and the fetch from the sinkhole at 805A, the final instruction is incorrectly generated as "jmp far ptr 0x10:0x8097". By placing a carefully crafted GDT at address 0, and configuring descriptor 0x10 to point outside of SMRAM, a payload at 0x8097 can intercept SMM execution to run with SMM privileges.

VII. Attack Payload

A prototype of the attack is provided in Figure 2, and has been validated on select processor models.

```
TARGET_SMBASE      equ 0x1f5ef800
GDT_ADDRESS        equ 0x10000
FJMP_OFFSET        equ 0x8097
DSC_OFFSET         equ 0xfb00
DESCRIPTOR_ADDRESS equ 0x10
APIC_BASE_MSR      equ 0x1b
SINKHOLE           equ ((TARGET_SMBASE+DSC_OFFSET)&0xfffff000)
PAYLOAD_OFFSET     equ 0x1000
CS_BASE            equ (PAYLOAD_OFFSET-FJMP_OFFSET)
APIC_BSP           equ 0x100
APIC_ACTIVE        equ 0x800

wbinvd
mov dword [dword GDT_ADDRESS+DESCRIPTOR_ADDRESS+4],
    (CS_BASE&0xff000000)|(0x00cf9a00)|(CS_BASE&0x00ff0000)>>16
mov dword [dword GDT_ADDRESS+DESCRIPTOR_ADDRESS+0],
    (CS_BASE&0x0000ffff)<<16|0xffff
mov eax, SINKHOLE | APIC_ACTIVE | APIC_BSP
mov edx, 0
mov ecx, APIC_BASE_MSR
wrmsr
jmp $
```

Fig. 2. A prototype sinkhole attack.

The attack is delivered through a kernel driver running in ring 0, and assumes an identity memory mapping. The example targets the BSP CPU core, on a system with SMBASE located at 0x1F5EF800. The GDT address is adjusted based on which APIC register is read in the GDT descriptor creation. The attack directs SMM execution to a secondary payload outside of SMRAM, at PAYLOAD_OFFSET. The secondary payload is installed by ring 0, and runs with SMM privileges, after the SMM handler is hijacked through the sinkhole. The specific effects of the secondary payload are left to the reader's imagination, but commonly include deeply persistent rootkits, hardware modifications, and system destruction.