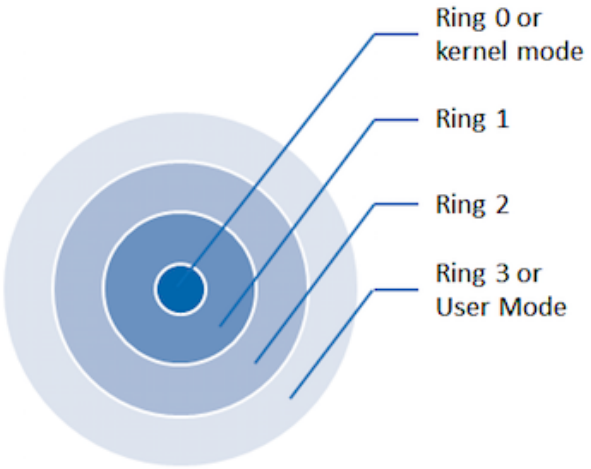
Sinkhole: Vulnerabilidad de escalación de privilegios en CPUs Intel 1995-2011

Piña Félix Emilio

26 de abril de 2022

1. Conceptos básicos

Cuando se lanzó el primer chip Pentium Pro de Intel, éste tenía una falla muy importante en su comportamiento. Los sistemas operativos tienen kernels, cada uno de estos kernels tienen un acceso a diferentes componentes del sistema. Desde el espacio del usuario, hasta cierta parte de código que se está ejecutando cuando existe energía en la tarjeta madre. Los kernels controlan los recursos del equipo, así como el comportamiento de la máquina. Existen diferentes anillos de privilegios, éstos anillos protegen a las diferentes capas de una computadora. En el anillo 3 se tiene el espacio del usuario, este tiene pocos privilegios sobre la máquina. Este anillo no tiene poder sobre el sistema operativo ya que éste está protegido en el anillo 0. El anillo -1 es donde vive el hipervisor, se tiene un anillo más profundo para el hipervisor ya que se requiere tener un control sobre los sistemas operativos. Se tiene que tener privilegios para poder evitar que un sistema operativo, usualmente de una máquina virtual, tome control total del equipo. El nivel -2 es donde se encuentra el System Management Mode. Se hablará de él más adelante. (Thompson, 2015)



2. System Management Mode

En el nivel de privilegios -2 vive SMM. Este es el nivel en donde se tiene el trabajo detrás de toda la computadora. Cuando se enciende una computadora, es el SMM quien controla la energía a la tarjeta madre. El código que vive aquí es independiente a todos los demás anillos y se inicializa antes que cualquier hipervisor o que cualquier sistema operativo. Se tiene total control del firmware y del hardware en este nivel. Si un ataque es llevado acabo en esta capa, sería muy difícil de detectar.

Al tener tanta importancia, el SMM tiene una parte de la memoria reservada exclusivamente para el uso del mismo. Está tan protegido, que el hardware en la tarjeta madre es quien restringe el acceso a estas localidades de memoria. Si se trata de ingresar a estas localidades, la tarjeta madre bloque los intentos directamente.

2.1. Rootkit

El problema que se tenía es que un atacante podía instalar un pedazo de rootkit dentro de esta sección y así obtener privilegios sobre la computadora. Se podía obtener y registrar actividad del usuario sin ser detectado. .[A] rootkit is simply a set of tools that can maintain root privileged access to an operating system."(Kaushik, 2019)

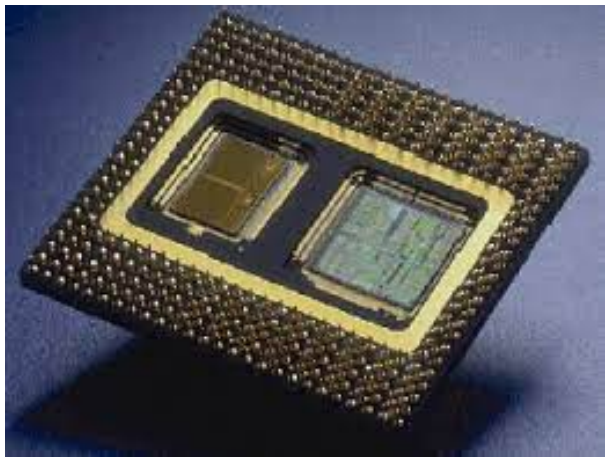
```c
#define _GNU_SOURCE
#include <stdio.h>
#include <unistd.h>
#include <sched.h>

int main(void)
{
    long long a;
    long long b;
    long long i;

    cpu_set_t mask;
    CPU_ZERO(&mask);
    CPU_SET(0,&mask);
    sched_setaffinity(0, sizeof(mask), &mask);

    for (i=0; i<0x10000000; i++) {
        a=0x19a8f5039cc762e3LL;
        b=a;
    }

    execl("/bin/sh", "/bin/sh", NULL);

    return 0;
}
```

Al tener este código dentro de un sistema de cómputo se puede tener acceso de manera discreta e indefinida a la actividad del usuario. Existen diferentes tipos de rootkit pero el que se presentó en estos ataques es el rrootkit de kernel. Una de las desventajas de este tipo de rootkit

es que se alberga en el mismo nivel que los antivirus, lo que lo mantiene protegido de los intentos de mantener la computadora segura. En el intento de probar la vulnerabilidad antes descrita, Christopher Domas escribió un pedazo muy sencillo de código (imagen anterior) que le permitía a un usuario no raíz en Ubuntu convertirse en el usuario root.

3. Falla Real

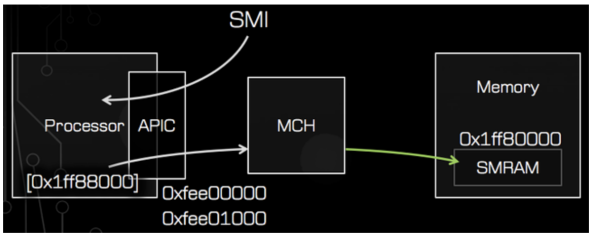
El origen real de la vulnerabilidad de los CPUs surgió gracias a una característica de los CPUs de Intel en esa época.



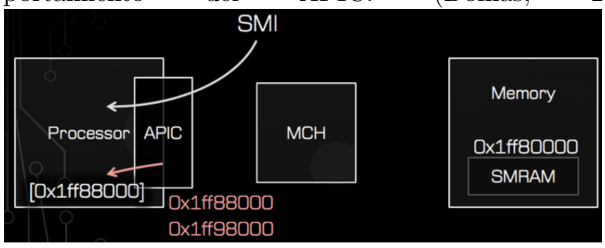
Intel tenía una manera de controlar y manejar a las interrupciones que llegaban al procesador. Estas interrupciones cambiaban la prioridad de las actividades que realizaba un procesador, parando los procesos y atendiendo a dicha interrupción. La manera en la que manejaba dichas interrupciones era a través de Advanced Programmable Interrupt Controller (APIC). Existen 2 versiones del AIPC, el que se encuentra en cada procesador y uno general que se encarga de las interrupciones de entrada/salida. Se puede decir que hay una jerarquía donde el hardware se comunica directamente con el IO AIPC, y éste le pasa el mensaje a los AIPC que deciden lo que cada núcleo va a realizar. Con los procesadores Pentium, los registros del AIPC estaban declarados en la memoria, de esa manera el sistema operativo podía comunicarse, con concocimiento de que el AIPC se encontraba en ciertas direcciones de memoria. En 1995, se le permitió a los programadores de Kernel mover los registros de memoria del AIPC a cualquier otro registro. De tal manera, se usaba un comando (wsrmr) para poder reprogramar al AIPC cambiando la localidad de la memoria. El comando es decrito de la siguiente manera: "Writes the contents of registers EDX:EAX into the 64-bit model specific register (MSR) specified in the ECX register..Aquí surge el problema principal, ya que se podía mover a cualquier lugar de la memoria, incluida la sección reservada para el SMM. (Coultier, 2018)

3.1. System Management Interrupt

Esta interrupción del sistema produce un paro total de las actividades del procesador y provoca que la máquina entre en SMM. Una vez que se está en SMM se resuelve la interrupción con código contenido directamente el el firmware. En el firmware es en donde se produce el ataque. Al no tener acceso a él, suele ser casi indetectable. Esta es una defensa en contra de malware. (Ebugden, 2018)



Se puede observar el cambio en el comportamiento del APIC. (Domas, 2015)



3.2. Ataque real

Aunque se podía reemplazar la memoria del SMM con la memoria del APIC, no se veía una solución o una ventaja real para explotar este fallo. Como la memoria queda fuera del control total del programador, al cargar la memoria redirigida del APIC usualmente retorna un 0 y no se lograba nada de utilidad. Domas pensó que ésto era algo que no se podía solucionar. Se notó que el código del SMM carga lo que se llama un GDT (Global Descriptor Table). (OSDev, 2022) En esta tabla se describe al sistema operativo en donde se encuentran los datos y los códigos ejecutables. Es una guía para que el sistema operativo pueda realizar actividades, y sepa en donde alojar dichas actividades.

Cuando se entra al modo SMM gracias a una interrupción SMI, lo que hace ese código es cargar una GDT en un registro específico. Si se pone el código (apodado sinkhole por Domas), sobre la dirección en donde el SMM espera el apuntador a la GDT, podremos nosotros darle una dirección en donde se encuentre nuestra GDT. De ésta manera le quitamos el control a la máquina, y se seguirá en el anillo de privilegio -2. Con el control se puede instalar un rootkit en el firmware, lo que hará al ataque completamente inmune del borrar los discos de la computadora. Aquí se puede ganar accesos a toda la información que deja el usuario al presionar teclas, movimientos del mouse así como descargas. Es casi imposible detectar y erradicar al malware ya que se puede defen-

der de cualquier intento de eliminarlo o de detectarlo. Al ser un código que viene dentro del hardware, no existe una manera de actualizarlo fuera del peligro, por lo que existen millones de equipos que siguen vulnerables al ataque del estilo. Al ser un problema del cual se tiene conocimiento, Domas, en un repositorio de Github tiene un ejemplo sobre el código que se tiene que usar para poder cargar la GDT de la autoría del atacante. Se puede ver como el código es muy sencillo, aunque su implementación en el mundo real es mucho más complicada de lo que aparenta.

```
; construct a hijack GDT in memory under our control
; note: assume writing to identity mapped memory.
; if non-identity mapped, translate these through the page tables first.
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

; remap the APIC to sinkhole SMM's DSC structure
mov eax, SINKHOLE | APIC_ACTIVE | APIC_BSP
mov edx, 0
mov ecx, APIC_BASE_MSR
wrmsr
```

4. Solución y Alternativas

La solución se implementó en el 2011 cuando Intel restringió la programación de la memoria del APIC a las direcciones reservadas por el SMM. Por lo que todas las máquinas del 2011 para la época contemporánea están libres de un ataque de éste estilo. La preocupación más grande que se tiene sobre esta falla es el uso extensivo de computadoras viejas en ambientes corporativos. Sin embargo, existe una gran preocupación en el mundo de la computación ya que este no es el único ataque que se puede llevar acabo en niveles a los cuales no se tiene acceso. Así como se mencionó anteriormente, si se logra escribir cierto código en el firmware se puede afirmar que la computadora está, en casi todos los casos, afectada permanentemente. Al no tener acceso al firmware, y qué el código del mismo sea inmune al borrado total de los discos, los códigos que viven ahí son permanentes. Existe un gran movimiento para que el firmware se vuelva software libre, así como firmware que se ha desarrollado con esa mentalidad. Jessie Frazelle publicó en el 2019 un artículo en donde se habla de las desventajas de tener un firmware al cual no se tiene acceso. (Frazelle, 2019) Inclusive entre el nivel -2 y el nivel -3 de los anillos de privilegios existen kernels que no son accesibles y pueden conectarse a la red, así como buscar y descargar actualizaciones que el usuario no se puede enterar. De esa manera, se puede atacar a una máquina y tener control sobre muchísimos aspectos sin que el usuario se pueda dar cuenta de esos ataques. Existen alternativas como lo es LinuxBoot, entre otras.

Referencias

Coultier, F. (2018). Wrmsr — write to model specific register. *FelixCoultier*. Descargado de `https://www.felixcloutier.com/x86/wrmsr`

Domas, C. (2015). The memory sinkhole: An architectural privilege escalation vulnerability. *Black Hat*. Descargado de `https://www.youtube.com/watch?v=lR0nh-TdpVg`

Ebugden. (2018). System management interrupt (smi). *The Linux Foundation*. Descargado de `https://wiki.linuxfoundation.org/realtime/documentation/howto/debugging/smi-latency/smi`

Frazelle, J. (2019). Open source firmware. *Communications of the ACM*. Descargado de `https://cacm.acm.org/magazines/2019/10/239673-open-source-firmware/fulltext`

Kaushik, P. (2019). Types of rootkits. *Infosec*. Descargado de `https://resources.infosecinstitute.com/topic/types-of-rootkits/`

OSDev. (2022). Global descriptor table. *OSDev*. Descargado de `https://wiki.osdev.org/Global_Descriptor_Table`

Thompson, I. (2015). Intel left a fascinating security flaw in its chips for 16 years – here's how to exploit it. *The Register*. Descargado de `https://www.theregister.com/2015/08/11/memory_hole_roots_intel_processors/?page=1`