Advanced x86:
BIOS and System Management Mode Internals
SMM & Caching

Xeno Kovah && Corey Kallenberg

LegbaCore, LLC

All materials are licensed under a Creative Commons “Share Alike” license.

http://creativecommons.org/licenses/by-sa/3.0/

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

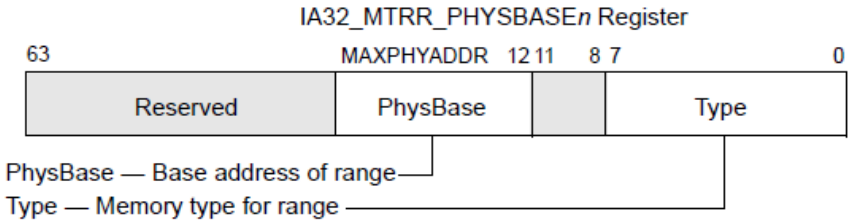
Attribution condition: You must indicate that derivative work
"Is derived from John Butterworth & Xeno Kovah’s ’Advanced Intel x86: BIOS and SMM’ class posted at http://opensecuritytraining.info/IntroBIOS.html”

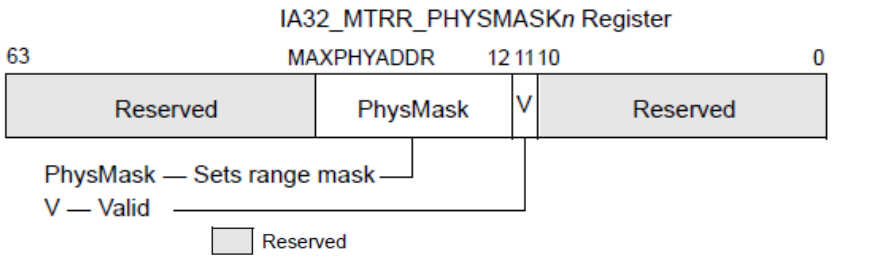
SMRAM and Caching

From Intel Vol. 3. Ch. "Memory Cache Control"

Cache Basics

- Temporary storage located on the CPU
- Accesses to data/instructions in cache are much faster than those to physical memory
- Caching is available in all operating modes, including SMM
- Caching type for a physical memory range is defined in Memory-Type Range Registers (MTRRs)
- MTRRs are a type of MSR (Model-Specific Register) that can be set to specify the type of CPU caching for ranges of physical memory
- Typically configured by BIOS but can also be configured by the operating system as needed





MAXPHYADDR: The bit position indicated by MAXPHYADDR depends on the maximum physical address range supported by the processor. It is reported by CPUID leaf function 80000008H. If CPUID does not support leaf 80000008H, the processor supports 36-bit physical address size, then bit PhysMask consists of bits 35:12, and bits 63:36 are reserved.

From Intel Vol. 3. Ch. "Memory Cache Control"

Memory Caching Types

- Physical memory ranges can be defined as having one of these types of caching properties
- The only one we'll discuss is the one that was the subject of the dual discovery by Duflot et al. and then later Wojtczuck et al.
  - Getting into SMRAM: SMM Reloaded, https://cansecwest.com/csw09/csw09-duflot.pdf
  - Attacking Memory via Intel CPU Cache Poisoning, http://invisiblethingslab.com/resources/misc09/smm_cache_fun.pdf
- The attack is brilliant in its simplicity

Table 11-8. Memory Types That Can Be Encoded in MTRRs

Memory Type and Mnemonic	Encoding in MTRR
Uncacheable (UC)	00H
Write Combining (WC)	01H
Reserved*	02H
Reserved*	03H
Write-through (WT)	04H
Write-protected (WP)	05H
Writeback (WB)	06H
Reserved*	7H through FFH

NOTE:

* Use of these encodings results in a general-protection exception (#GP).

Write-back (WB)

- The point of Write-back caching is to reduce the amount of bus traffic between the processor and memory
- Reads come from cache lines on cache hits
- Writes are performed in the cache and not immediately written/flushed to memory
- Both read and write *misses* cause cache fills
- Modified CPU cache lines are written back (write-back) to memory at a later time*

- Simply put, reading/writing from/to a memory region that uses write-back caching will initially fill a line in the CPU cache
- Subsequent reads/writes from/to that address will be from/to cache instead of memory
- Until the processor writes-back that cache to memory*

*Read the Intel Software Developers Guide Volume 3

https://cansecwest.com/csw09/csw09-duflot.pdf

So…

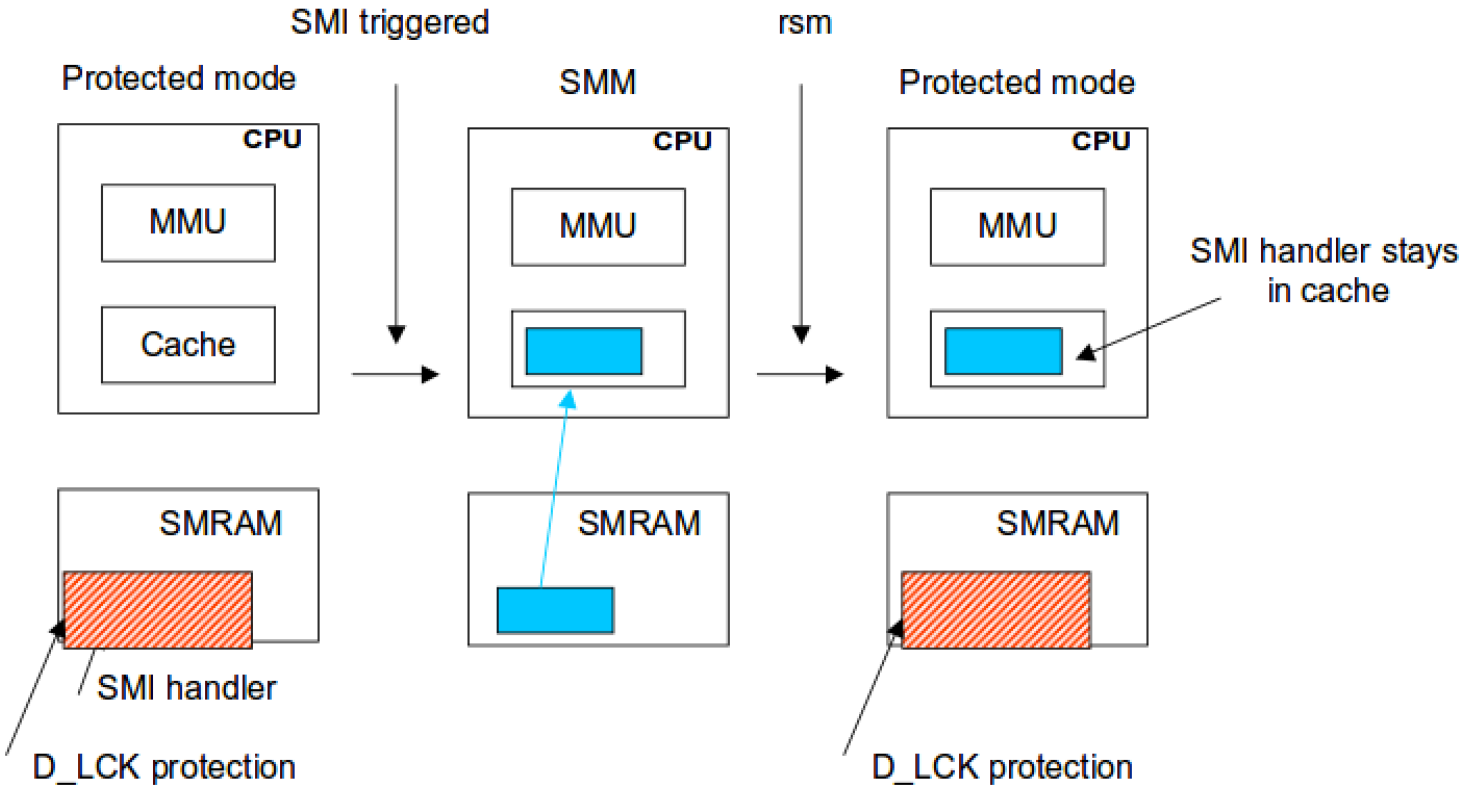
(D_LCK bit)

- The access control point is in the chipset and the chipset does not "see" what happens inside the cache.
- Code running on the CPU can decide the caching strategy.
- Plus, the chipset does not even know where the SMRAM really is (SMBASE only known to the CPU).

- Isn't there a coherency problem here?

https://cansecwest.com/csw09/csw09-duflot.pdf

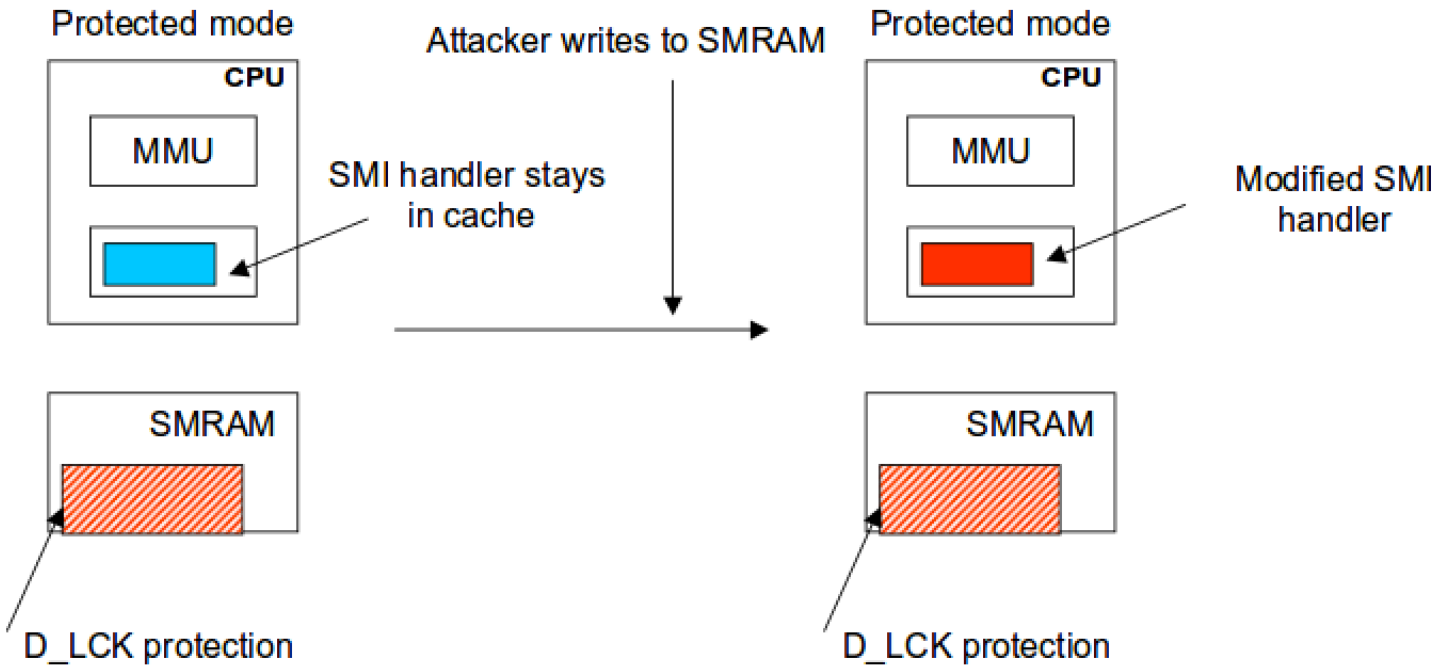
Basic idea: SMI handler stays in cache



https://cansecwest.com/csw09/csw09-duflot.pdf

Basic idea: attacker writes to the SMRAM

- When the CPU is not in SMM, the CPU cannot write in SMRAM. But if the SMRAM is cached in Write Back mode, the CPU only writes to the cached version and not in memory.

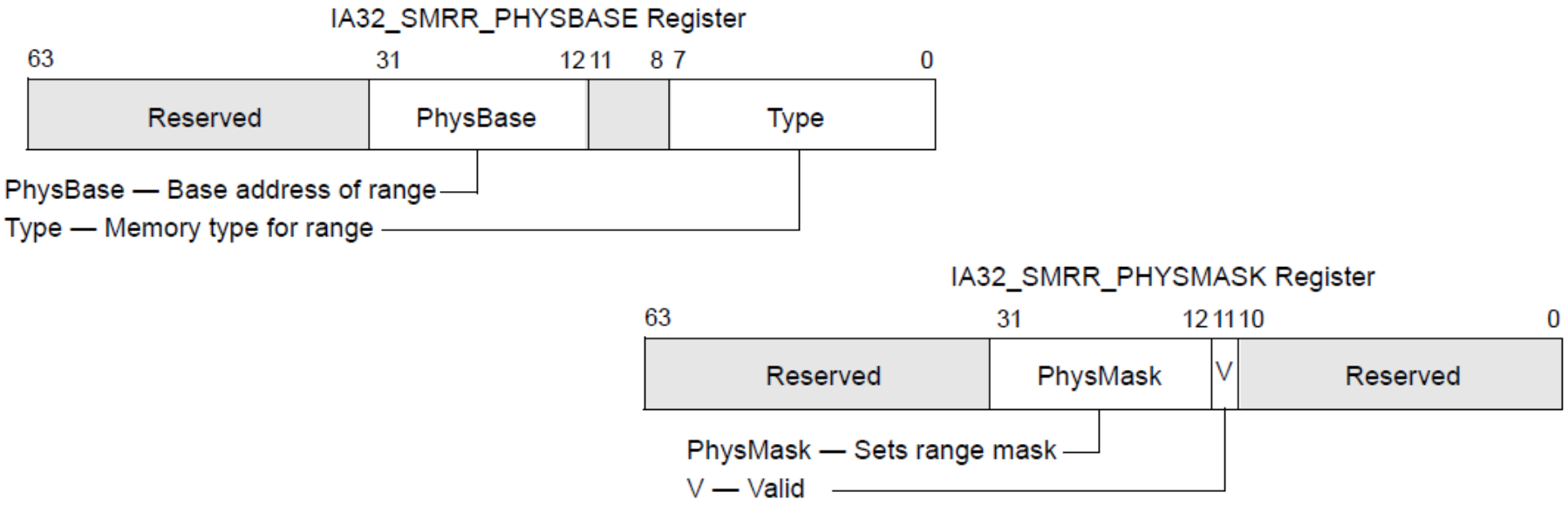


The fix: SMRR

- The preceding is a great example of how security researchers can influence industry for the better. Damn fine job.
- System-Management Range Register (SMRR) was introduced in Intel's x64 architecture*
- Provides a PHYSBASE/PHYSMASK pair just like MTRRs
- Prevents the kind of attack that we just saw in the preceding example
- SMRR restricts access to the address range defined in the SMRR registers
- Defines the memory type (caching) for the SMRAM range
- *SMRRs can be written to only when the processor is in SMM*
- SMRR takes priority over MTRR in case of overlapping ranges

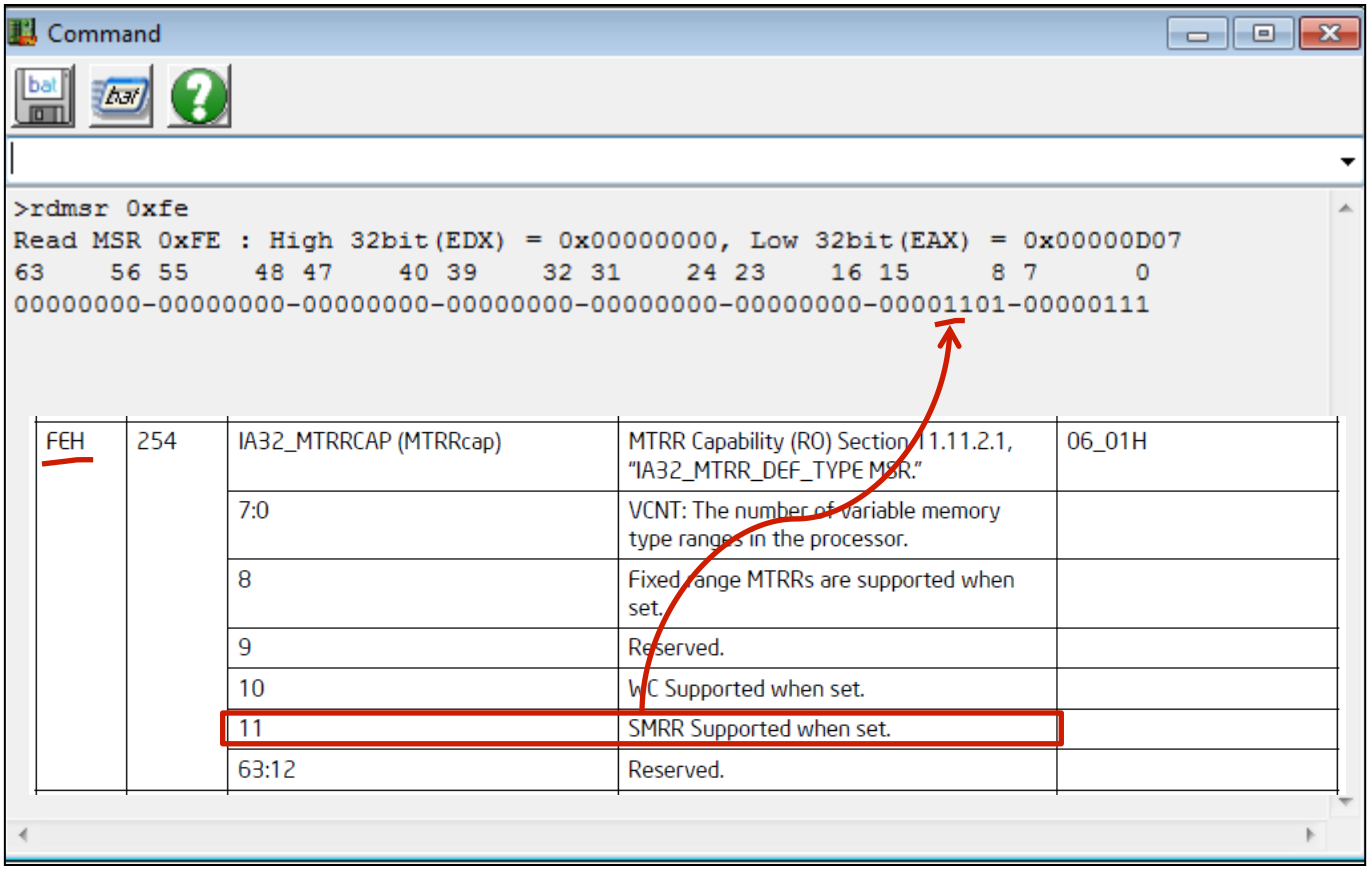
* This is one of the only architecture-dependent security mechanisms. So far up to this point all has been x32/x64 agnostic

SMRR



- When the processor is in SMM:
  - Memory accesses to this range will use the memory type defined in SMRR_PHYSBASE
- When the processor is not in SMM:
  - Memory reads return a fixed value (0xFF in my experience)
  - Memory writes are ignored
  - Memory type is Uncacheable

Verify SMRR Support: IA32_MTRRCAP



FEH	254	IA32_MTRRCAP (MTRRcap)	MTRR Capability (RO) Section 11.11.2.1, "IA32_MTRR_DEF_TYPE MSR."	06_01H
		7:0	VCNT: The number of variable memory type ranges in the processor.	
		8	Fixed range MTRRs are supported when set.	
		9	Reserved.	
		10	WC Supported when set.	
		11	SMRR Supported when set.	
		63:12	Reserved.	

- SMRR is supported on a system if bit 11 in the IA32_MTRRCAP MSR is set
- Verify next that it is being used

SMRR MSR Number

A0H	160	MSR_SMRR_PHYSBASE	Unique
		11:0	
		31:12	
		63:32	

For the reference E6400 (Core2Duo)

1F2H	498	IA32_SMRR_PHYSBASE
		7:0
		11:8
		31:12
		63:32

- If you try to read the SMRR of your system, be sure to verify its location using the developers guide (MSR chapter)
- The MSR register addresses are non "architectural" and will therefore differ between architectures
  - That's why they are called Model-Specific Registers
- RW-E does not appear to handle exceptions well since reading the wrong MSR will crash your system
  - As of latest version

Homework heads up

- Find the value of SMRR_PHYSBASE for your particular hardware