Advanced x86:
BIOS and System Management Mode Internals
SPI Flash Protection Mechanisms

Xeno Kovah && Corey Kallenberg

LegbaCore, LLC

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

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

You are free:

to Share — to copy, distribute and transmit the work

to Remix — to adapt the work

Under the following conditions:

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

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

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

How to stop someone from writing to your BIOS

- AKA “How to stop an attacker from writing to your BIOS”
  - AKA “What the BIOS vendors are typically configuring wrong when they’re supposed to be stopping attackers from writing to your BIOS”

Flash Protection

Table 5-60. Flash Protection Mechanism Summary

Mechanism	Accesses Blocked	Range Specific?	Reset-Override or SMI#-Override?	Equivalent Function on FWH
BIOS Range Write Protection	Writes	Yes	Reset Override	FWH Sector Protection
Write Protect	Writes	No	SMI# Override	Same as Write Protect in previous ICHs for FWH

- With no Flash Descriptor present, the only mechanisms to lock the flash are:
  - BIOS Range Write Protection
  - Global Flash Write Protection
- The reference to FWH Sector Protection yields no results in any datasheets. I am assuming it is related to the R/W control shown in the sample FWH register map at the beginning of the BIOS Flash section

Flash Protection Mechanism #1

Table 5-60. Flash Protection Mechanism Summary

Mechanism	Accesses Blocked	Range Specific?	Reset-Override or SMI#-Override?	Equivalent Function on FWH
BIOS Range Write Protection	Writes	Yes	Reset Override	FWH Sector Protection
Write Protect	Writes	No	SMI# Override	Same as Write Protect in previous ICHs for FWH

- Covering this first because I believe it to be your first line of defense to protect your BIOS flash from writes
- Applies to the entire flash chip (Global Flash Protection)
- Provides SMM the ability to determine whether or not a request to unlock the BIOS flash for writing will be permitted
- This protection is provided by the chipset (not on the flash itself)

Global BIOS Write Protection

BIOS_CNTL—BIOS Control Register (LPC I/F—D31:F0)

Offset Address:	DCh	Attribute:	R/WLO, R/W, RO
Default Value:	00h	Size:	8 bit
Lockable:	No	Power Well:	Core

Bit	Description
7:5	Reserved
4	**Top Swap Status (TSS)** — RO. This bit provides a read-only path to view the state of the Top Swap bit that is at offset 3414h, bit 0.
3:2	**SPI Read Configuration (SRC)** — R/W. This 2-bit field controls two policies related to BIOS reads on the SPI interface: Bit 3- Prefetch Enable Bit 2- Cache Disable Settings are summarized below: **Bits 3:2** — **Description** 00b — **No prefetching, but caching enabled.** 64B demand reads load the read buffer cache with "valid" data, allowing repeated code fetches to the same line to complete quickly 01b — **No prefetching and no caching.** One-to-one correspondence of host BIOS reads to SPI cycles. This value can be used to invalidate the cache. 10b — **Prefetching and Caching enabled.** This mode is used for long sequences of short reads to consecutive addresses (i.e., shadowing). 11b — **Reserved. This is an invalid configuration**, caching must be enabled when prefetching is enabled.
1	**BIOS Lock Enable (BLE)** — R/WLO. 0 = Setting the BIOSWE will not cause SMIs. 1 = Enables setting the BIOSWE bit to cause SMIs. Once set, this bit can only be cleared by a PLTRST#
0	**BIOS Write Enable (BIOSWE)** — R/W. 0 = Only read cycles result in Firmware Hub I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

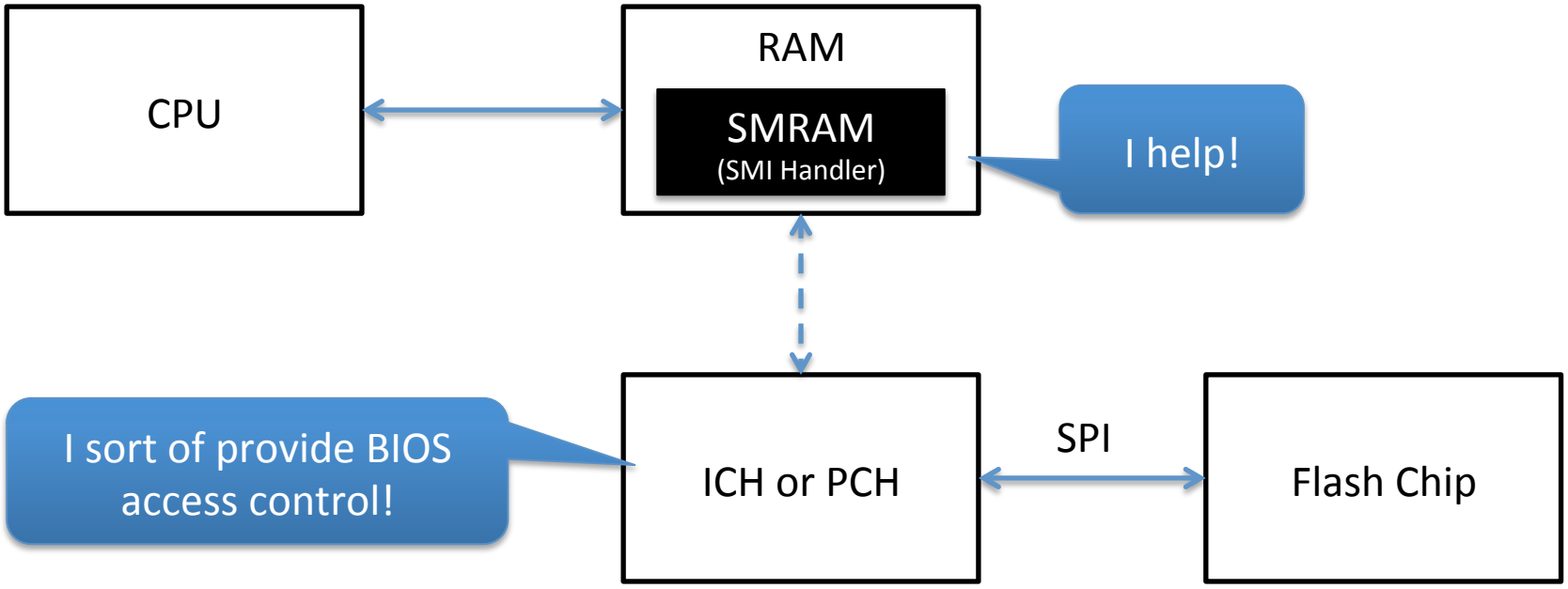
- On MCH/ICH systems, bits 7:5 of the BIOS_CNTL are reserved
- On this system BIOS_CNTL is located in the LPC device (D31:F0, offset DCh)
- These protections would also apply to the Firmware Hub (FWH) if the BIOS were located there.

ICH/PCH Chipset SMM-derived Write Protection:

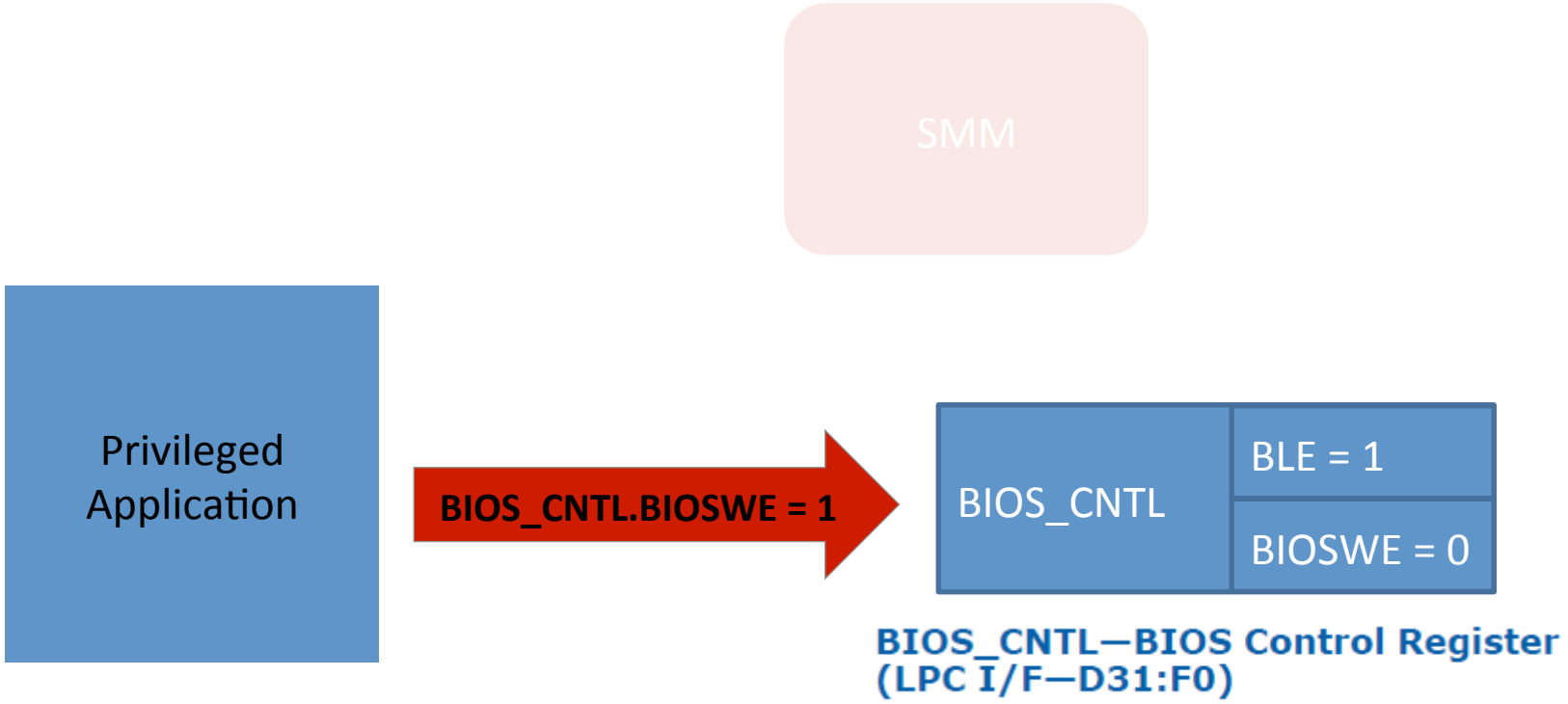
1	**BIOS Lock Enable (BLE)** — R/WLO. 0 = Setting the BIOSWE will not cause SMIs. 1 = Enables setting the BIOSWE bit to cause SMIs. Once set, this bit can only be cleared by a PLTRST#
0	**BIOS Write Enable (BIOSWE)** — R/W. 0 = Only read cycles result in Firmware Hub I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

- BIOS_CNTL.BIOSWE (bit 0) enables write access to the flash chip
  - Always R/W
- BIOS_CNTL.BLE (bit 1) provides an opportunity for the OEM to implement an SMI to protect the BIOSWE bit

How you should think of BLE

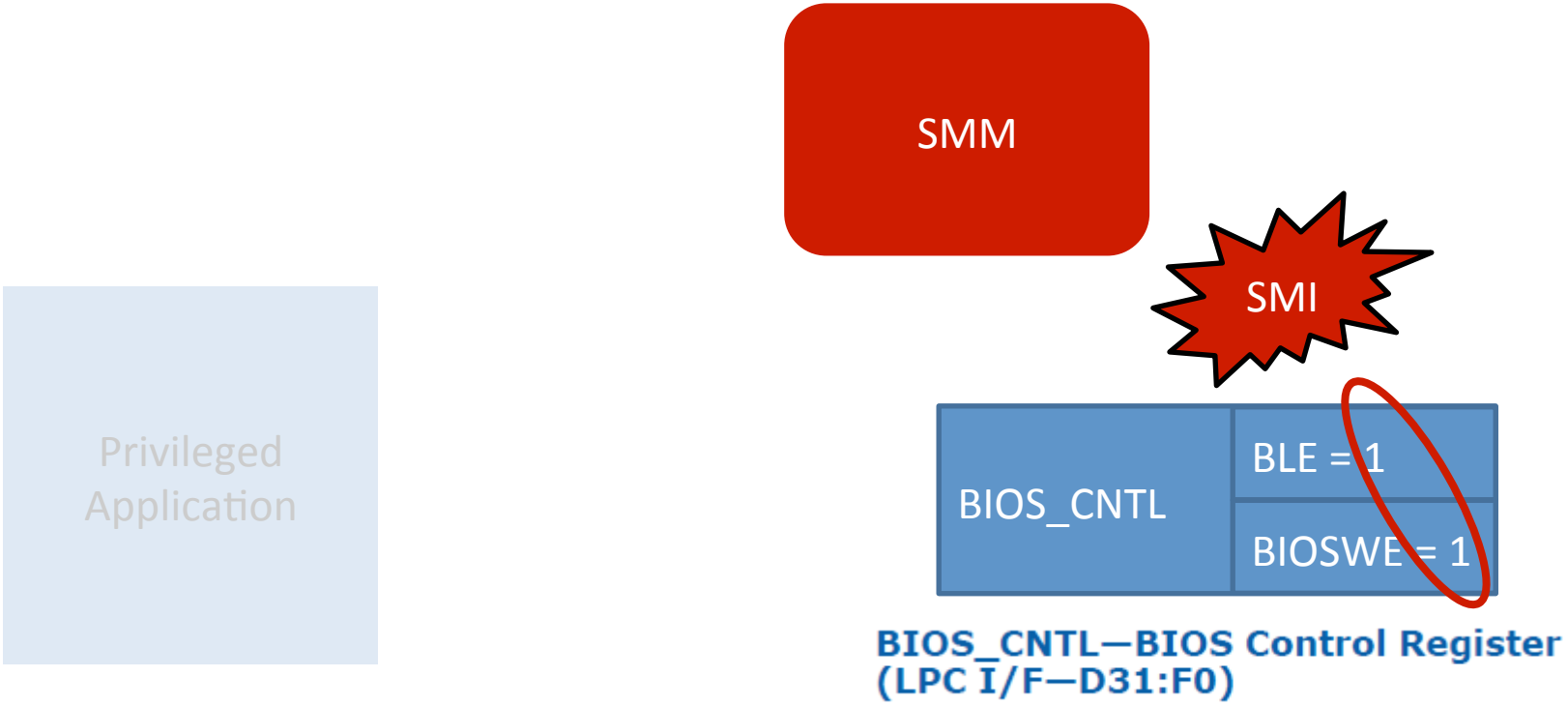


How it works



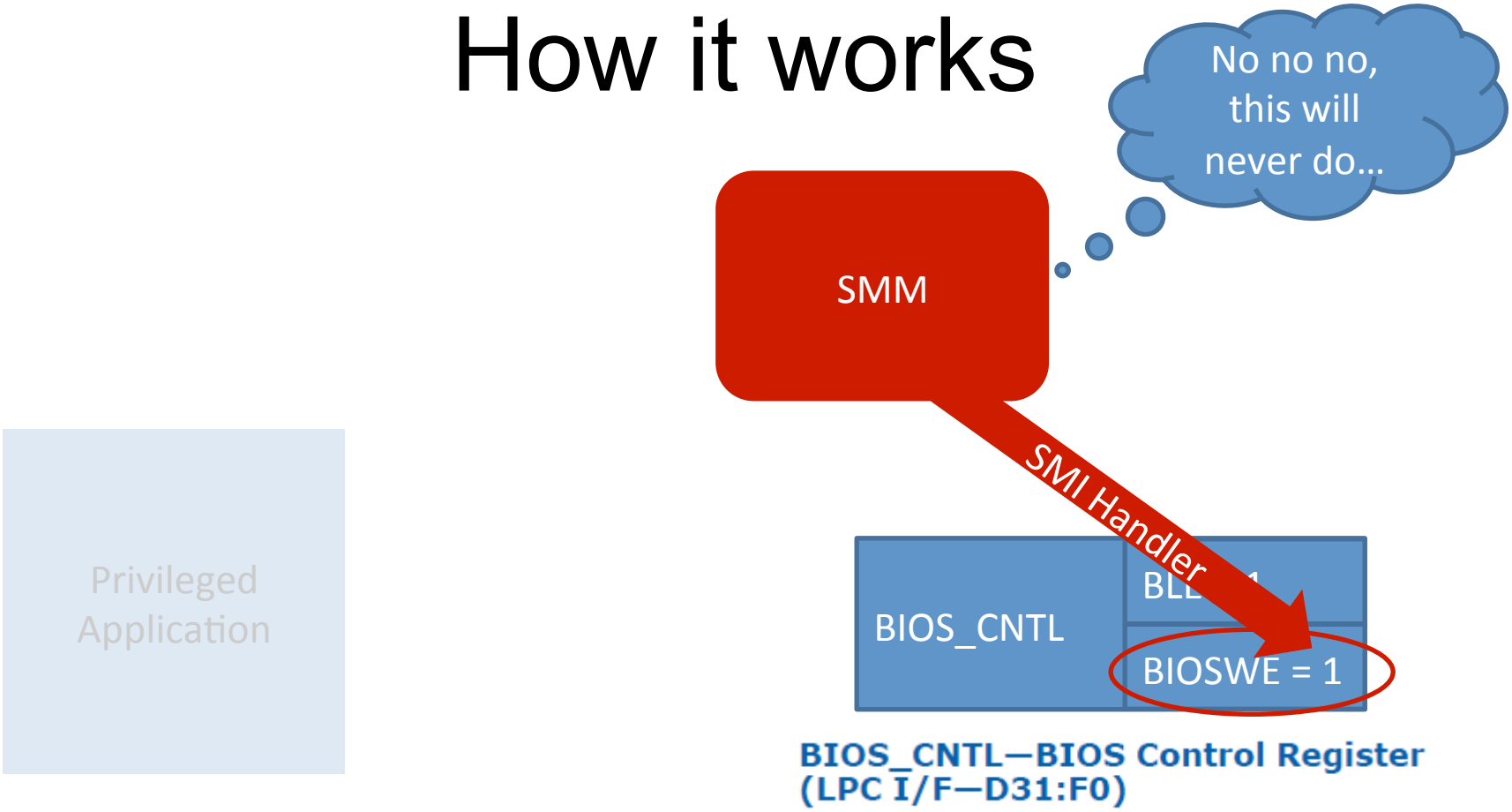
- Privileged app wants to write to the SPI flash, sets BIOS_CNTL.BIOSWE to 1
  - The only reason privileges are needed is to execute the in/out instructions

How it works



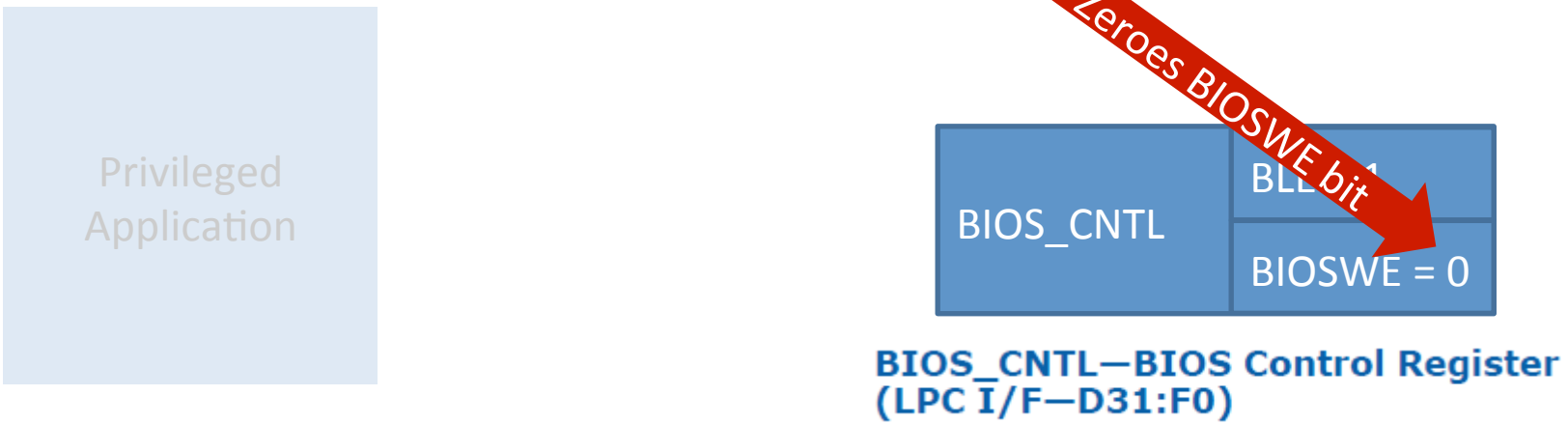
- The BIOS_CNTL register has the BIOS Lock (BLE) enabled
- Asserting BIOSWE while BLE is set generates an SMI#
  - SMI# is initiated by the Chipset (ICH)
- The processor transitions to System Management Mode

How it works



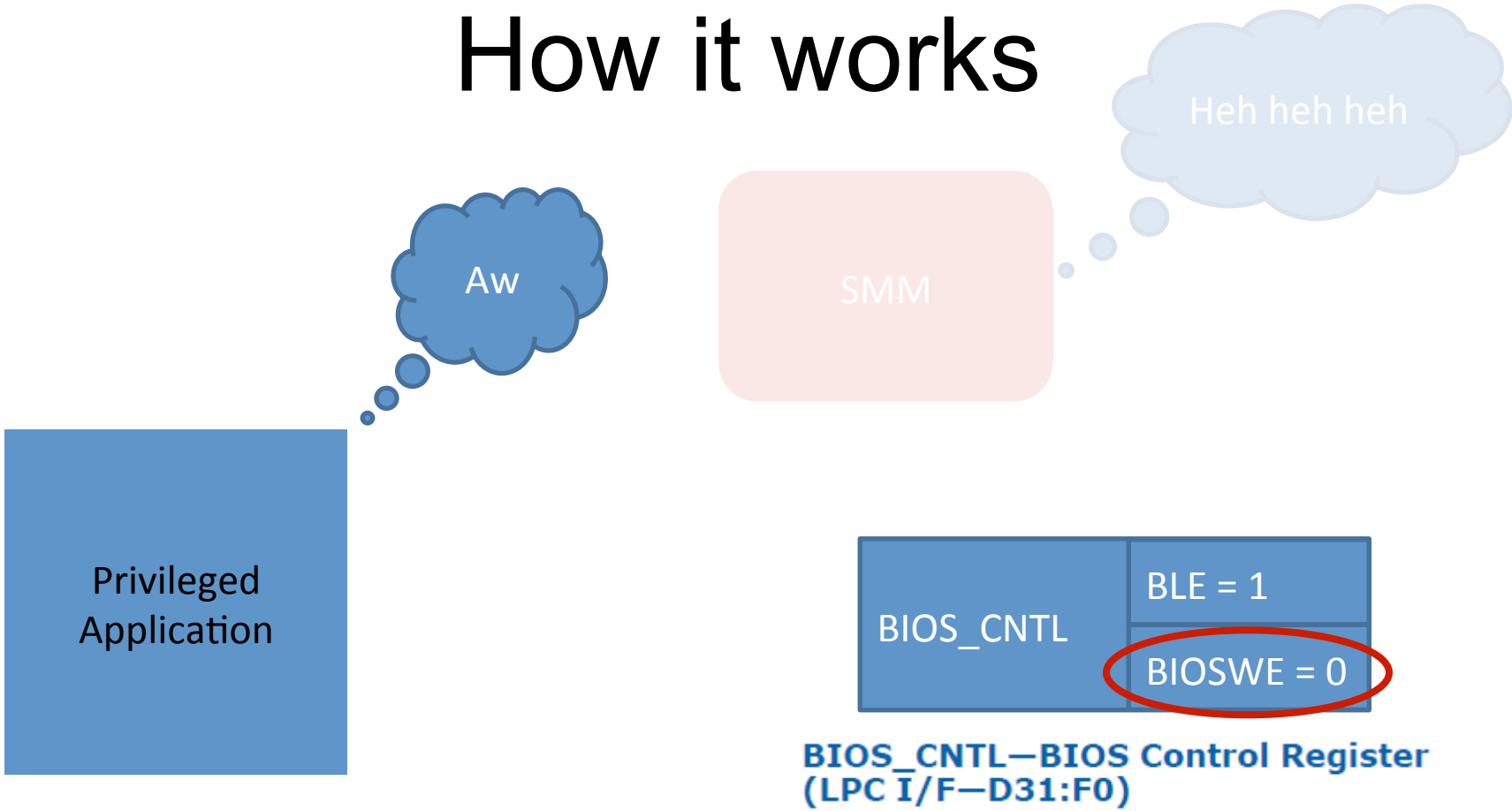
- A routine in the SMI handler explicitly checks to see if BIOS_CNTL.BIOSWE is set

How it works



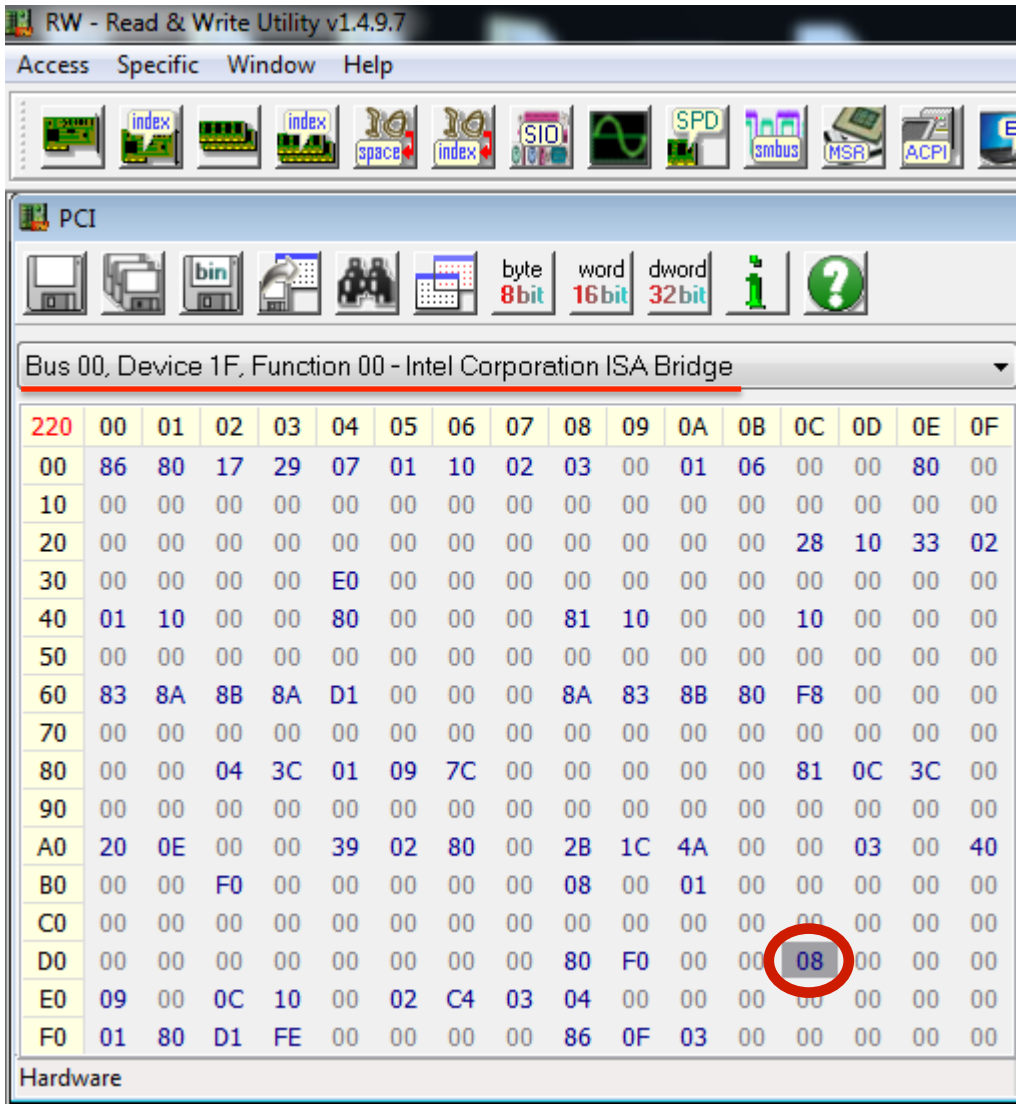
- The SMI handler flips this bit back to 0, disabling writes to the serial flash
  - Since updates should be applied only by SMRAM, SMRAM knows that unless it flipped this bit, this bit shouldn't be flipped.

How it works



- From the app's perspective, it appears the BIOSWE bit was never even asserted.
- Of course this only works if:
  - BLE is asserted/enabled
  - There is a SMI handler explicitly checking/resetting the BIOSWE bit
  - SMIs cannot be somehow suppressed (you already saw 1 way)

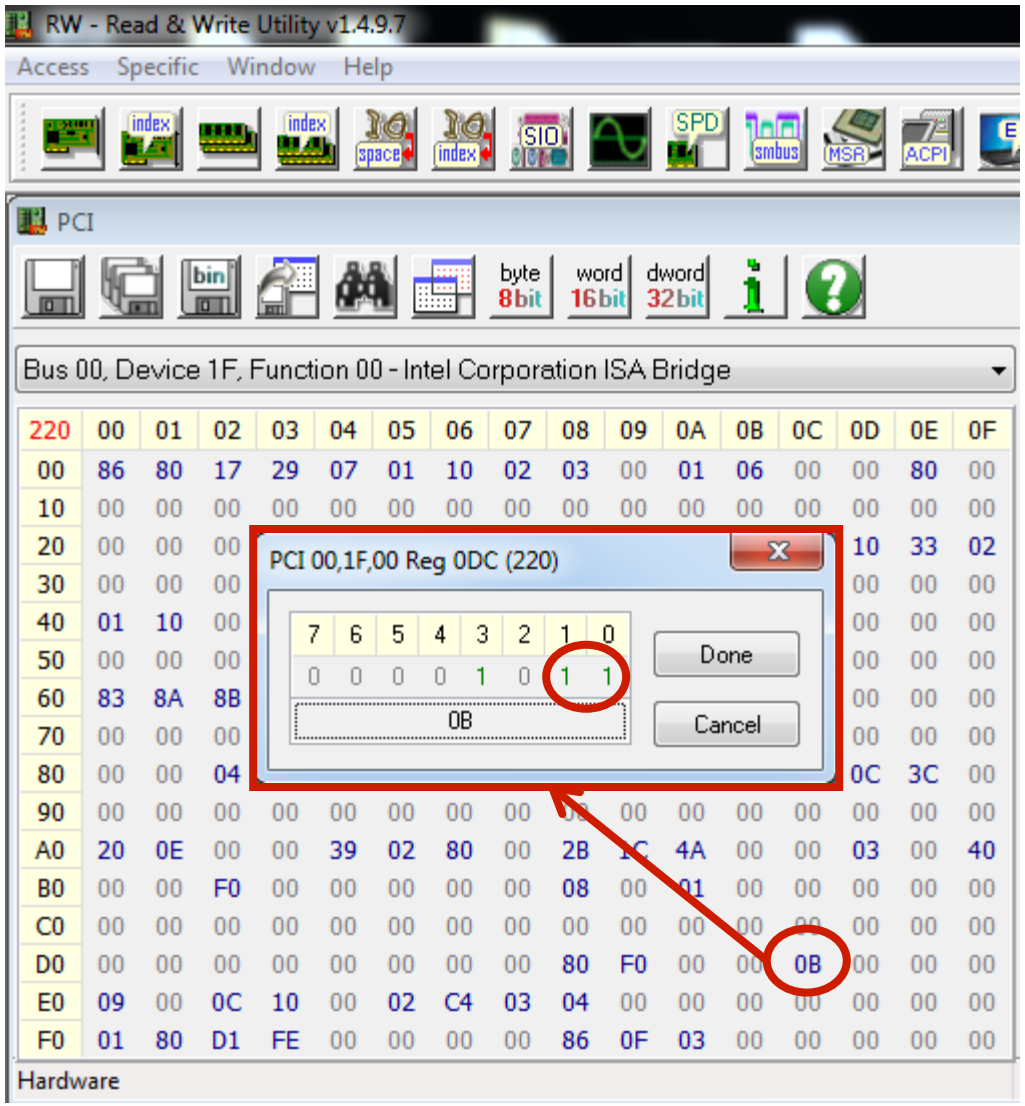
vulnBIOS example: BIOS_CNTL Testing



220	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	86	80	17	29	07	01	10	02	03	00	01	06	00	00	80	00
10	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
20	00	00	00	00	00	00	00	00	00	00	00	00	28	10	33	02
30	00	00	00	00	E0	00	00	00	00	00	00	00	00	00	00	00
40	01	10	00	00	80	00	00	00	81	10	00	00	10	00	00	00
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	83	8A	8B	8A	D1	00	00	00	8A	83	8B	80	F8	00	00	00
70	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
80	00	00	04	3C	01	09	7C	00	00	00	00	00	81	0C	3C	00
90	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
A0	20	0E	00	00	39	02	80	00	2B	1C	4A	00	00	03	00	40
B0	00	00	F0	00	00	00	00	00	08	00	01	00	00	00	00	00
C0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
D0	00	00	00	00	00	00	00	00	80	F0	00	00	08	00	00	00
E0	09	00	0C	10	00	02	C4	03	04	00	00	00	00	00	00	00
F0	01	80	D1	FE	00	00	00	00	86	0F	03	00	00	00	00	00

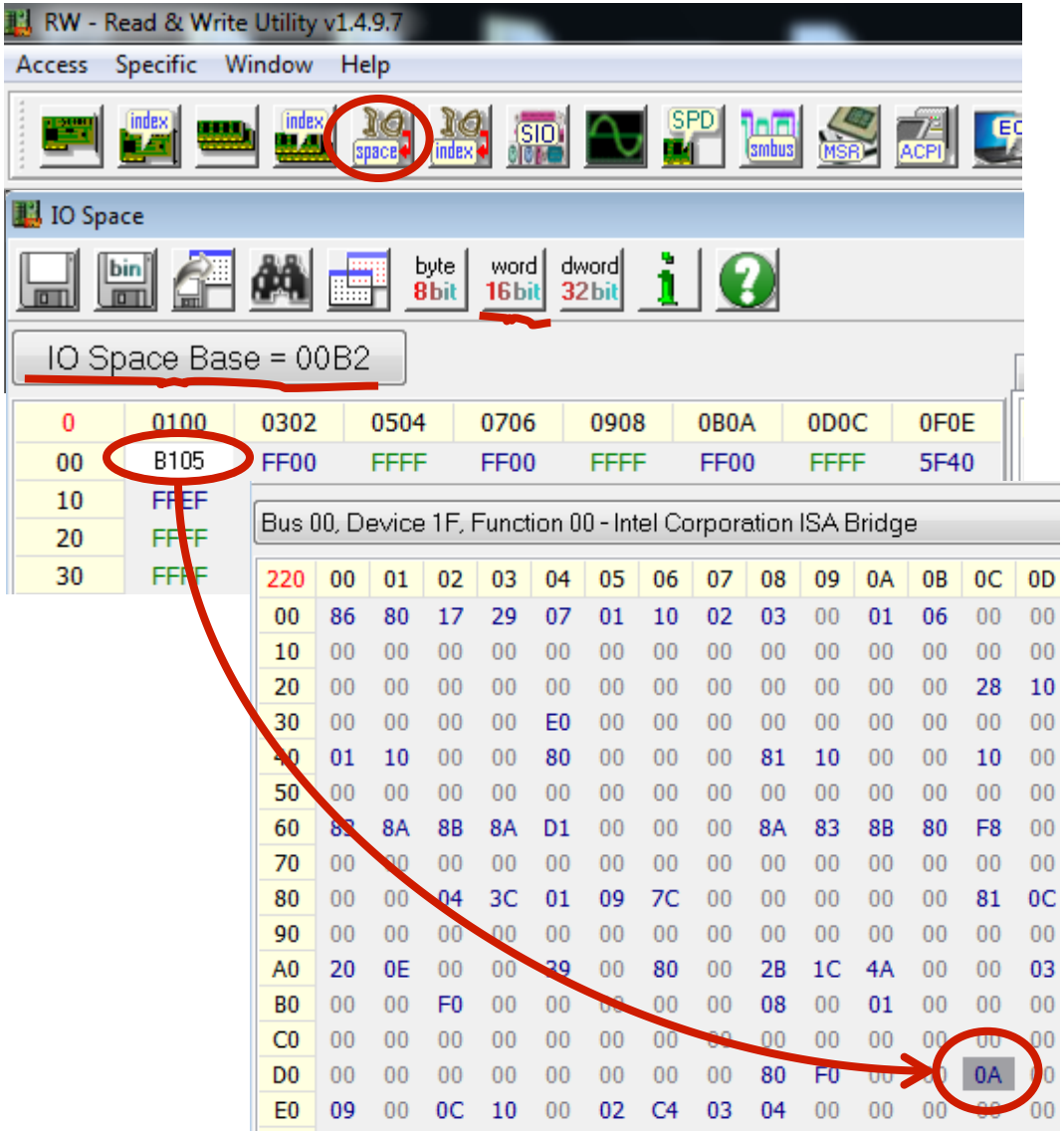
- Look at the BIOS_CNTL register in the LPC device
- BIOS Lock Enable (BLE) bit 1 is not asserted
- This means any application privileged enough to either map the PCI Express configuration space or perform port I/O can assert BIOSWE to enable writes to the BIOS flash

vulnBIOS example: BIOS_CNTL Testing



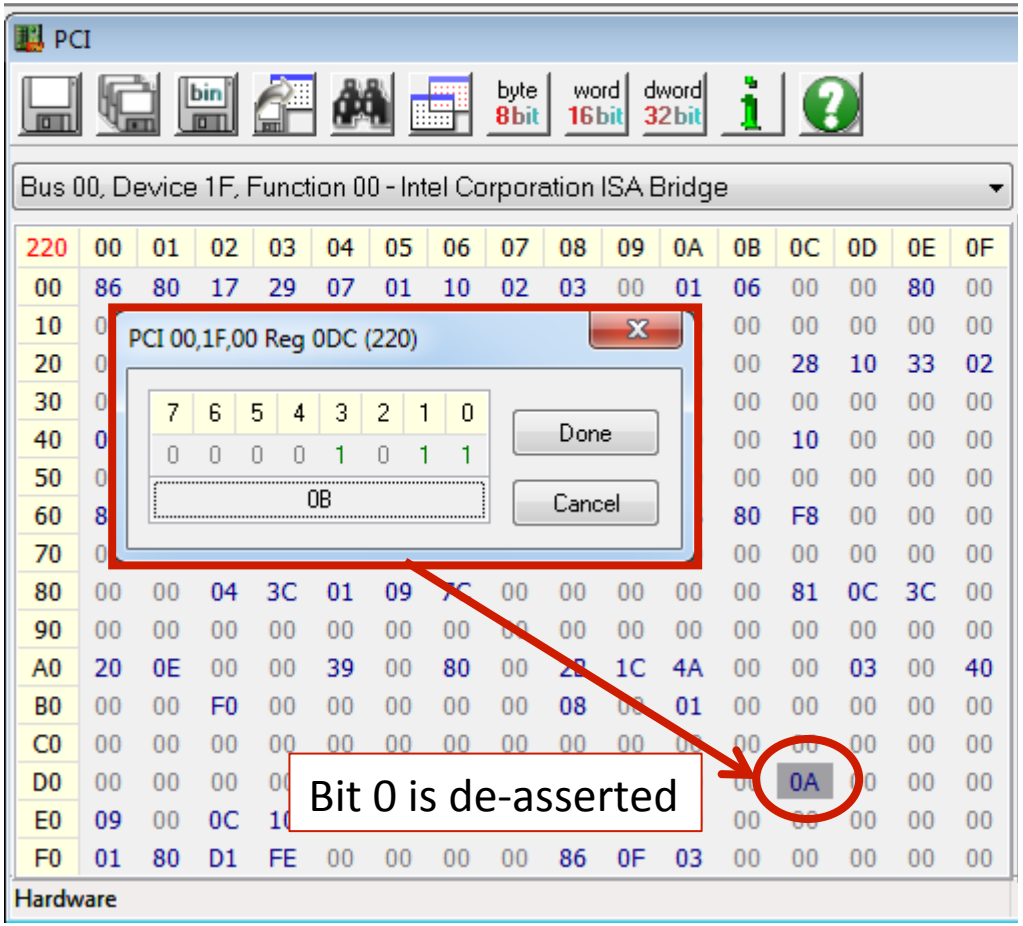
- You don't have to do this, but note that it is possible to set BIOS Lock Enable to 1 and not have an SMI handler routine running that checks and de-asserts BIOS Write Enable bit 0
- We've seen this on multiple systems; where BLE was set, but asserting BIOSWE to 1 was not reset to 0
- This is why bit 0 must be tested in order to really test write-protection

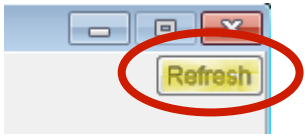
vulnBIOS example: BIOS_CNTL Testing: Set BLE



- Now we're going to setup BIOS_CNTL so that it protects the flash
- This will only work on these lab machines with this modified BIOS
- I inserted a custom SMI handler that resets the BIOSWE bit to 0 if it is asserted
- We're going to enable this by writing the word 0xb105 to port 0xB2
- You should now see the BLE bit asserted in BIOS_CNTL (0x0A)

vulnBIOS example: BIOS_CNTL Testing





- Now try to enable writes to the BIOS by asserting BIOSWE bit 0
  - Set BIOS_CNTL to 0x0B
- You will notice that it resets to 0x0A
- This is the SMI handler working as it should
- Note the reset of BIOSWE to 0 occurs during SMM
- Any tangible delay you see in resetting this value is due to the (configurable) “Refresh” button in RW-E

BIOSWE/BLE should be considered deprecated!

- We can defeat it on systems that are not using SMRRs
  - "The Sicilian" – "Defeating Signed BIOS enforcement", Kallenberg et al., EkoParty 2013
- We can defeat it on systems that don't set SMI_LOCK
  - "Charizard" – "Setup for Failure: Defeating UEFI Secure Boot", Kallenberg et al., Syscan 2014
  - But Charizard actually found by Sam Cornwell, it just got merged into Corey's talk in its first appearance. Will be spun off later.
- We can defeat it on systems TXT-enabled that suppress SMIs
  - "Sandman" – "SENTER Sandman: Using Intel TXT to attack BIOSes", Kovah et al., Summercon 2014
- We have a new fundamental attack against it that will bypass BLE *on all systems*, once and for all. “Speed Racer” We’ll talk about these at the end, depending on time.

BIOS_CNTL: SMM_BWP

BIOS_CNTL—BIOS Control Register (LPC I/F—D31:F0)

Offset Address:	DCh	Attribute:	R/WLO, R/W, RO
Default Value:	20h	Size:	8 bits
Lockable:	No	Power Well:	Core

Bit	Description
7:6	Reserved
5	**SMM BIOS Write Protect Disable (SMM_BWP)**—R/WL. This bit set defines when the BIOS region can be written by the host. 0 = BIOS region SMM protection is disabled. The BIOS Region is writable regardless if processors are in SMM or not. (Set this field to 0 for legacy behavior). 1 = BIOS region SMM protection is enabled. The BIOS Region is not writable unless all processors are in SMM and BIOS Write Enable (BIOSWE) is set to '1'.
4	**Top Swap Status (TSS)**—RO. This bit provides a read-only path to view the state of the Top Swap bit that is at offset 3414h, bit 0.
3:2	**SPI Read Configuration (SRC)**—R/W. This 2-bit field controls two policies related to BIOS reads on the SPI interface: Bit 3 – Prefetch Enable Bit 2 – Cache Disable Settings are summarized below: **Bits 3:2** — **Description** 00b — **No prefetching, but caching enabled.** 64B demand reads load the read buffer cache with "valid" data, allowing repeated code fetches to the same line to complete quickly. 01b — **No prefetching and no caching.** One-to-one correspondence of host BIOS reads to SPI cycles. This value can be used to invalidate the cache. 10b — **Prefetching and Caching enabled.** This mode is used for long sequences of short reads to consecutive addresses (that is, shadowing).
1	**BIOS Lock Enable (BLE)**—R/WLO. 0 = Transition of BIOSWE from '0' to '1' will not cause an SMI to be asserted. 1 = Enables setting the BIOSWE bit to cause SMIs and locks SMM_BWP. Once set, this bit can only be cleared by a PLTRST#.
0	**BIOS Write Enable (BIOSWE)**—R/W. 0 = Only read cycles result in Firmware Hub or SPI I/F cycles. 1 = Access to the BIOS space is enabled for both read and write cycles. When this bit is written from a 0 to a 1 and BIOS Lock Enable (BLE) is also set, an SMI# is generated. This ensures that only SMI code can update BIOS.

- In PCH chipsets, bit 5 of BIOS_CNTL has been defined:
- Provides the vendor the ability to ensure that BIOS region may ONLY be written to when all processors are in SMM and BIOSWE is enabled
- Our lab system does not implement this bit because it is an MCH/ICH system, but check it out on your own
- As we've seen, this register is important to lock down the BIOS to mitigate SMI suppression
- Same here
- Only 6 out of ~10k systems we've measured to date use it!!! ☹
  - As of 3/31/2014

Another Protection Mechanism

Table 5-60. Flash Protection Mechanism Summary

Mechanism	Accesses Blocked	Range Specific?	Reset-Override or SMI#-Override?	Equivalent Function on FWH
BIOS Range Write Protection	Writes	Yes	Reset Override	FWH Sector Protection
Write Protect	Writes	No	SMI# Override	Same as Write Protect in previous ICHs for FWH

- BIOS Range Write-Protection is the second major line of defense
- There are 5 Protected Range registers (0-4) with independent R/W permissions
- Setting these will prevent reads and/or writes until the system is reset.

* Information on FWH Sector Protection is hard to come by. It appears to be a security mechanic on the chip itself, since chips can be described as being divided into sectors.

PR0—Protected Range 0 Register (SPI Memory Mapped Configuration Registers)

Memory Address: [illegible] + 74h
Default Value: 00000000h

Attribute: R/W
Size: 32 bits

So who protects the protector?

This guy that's who! We'll get to that later

This register can not be written when the FLOCKDN bit is set to 1.

Bit	Description
31	**Write Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that writes and erases directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
30:29	Reserved
28:16	**Protected Range Limit** — R/W. This field corresponds to FLA address bits 24:12 and specifies the upper limit of the protected range. Address bits 11:0 are assumed to be FFFh for the limit comparison. Any address greater than the value programmed in this field is unaffected by this protected range.
15	**Read Protection Enable** — R/W. When set, this bit indicates that the Base and Limit fields in this register are valid and that read directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
14:13	Reserved
12:0	**Protected Range Base** — R/W. This field corresponds to FLA address bits 24:12 and specifies the lower base of the protected range. Address bits 11:0 are assumed to be 000h for the base comparison. Any address less than the value programmed in this field is unaffected by this protected range.

Example of a Protect Range Register

Protected Range (PR) Registers

- The protections prescribed herein are enforced even upon the SMI handler
- Enforced on register access, not direct access, however
- Protected ranges are available to a SPI flash operating in either Non-Descriptor mode or Descriptor mode
  - The ranges don't have to mirror descriptor mode regions
- Base addresses must be page-aligned
  - Lower 12 bits are 000h
- Limit addresses end at one under a page aligned boundary
  - Lower 12 bits are FFFh
- Addresses are Flash Linear Addresses (FLAs)
  - Basically an offset from the base of the flash
  - So "offset" 0x260000 on the flash is Flash Linear Address 0x260000

PR Sample: 03FF02A2h
Base 2A2000h

Protected Range Base

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
WP			0	0	0	1	1	1	1	1	1	1	1	1	1	RP			0	0	0	1	0	1	0	1	0	0	0	1	0

FLA_{base} = 2A2000h

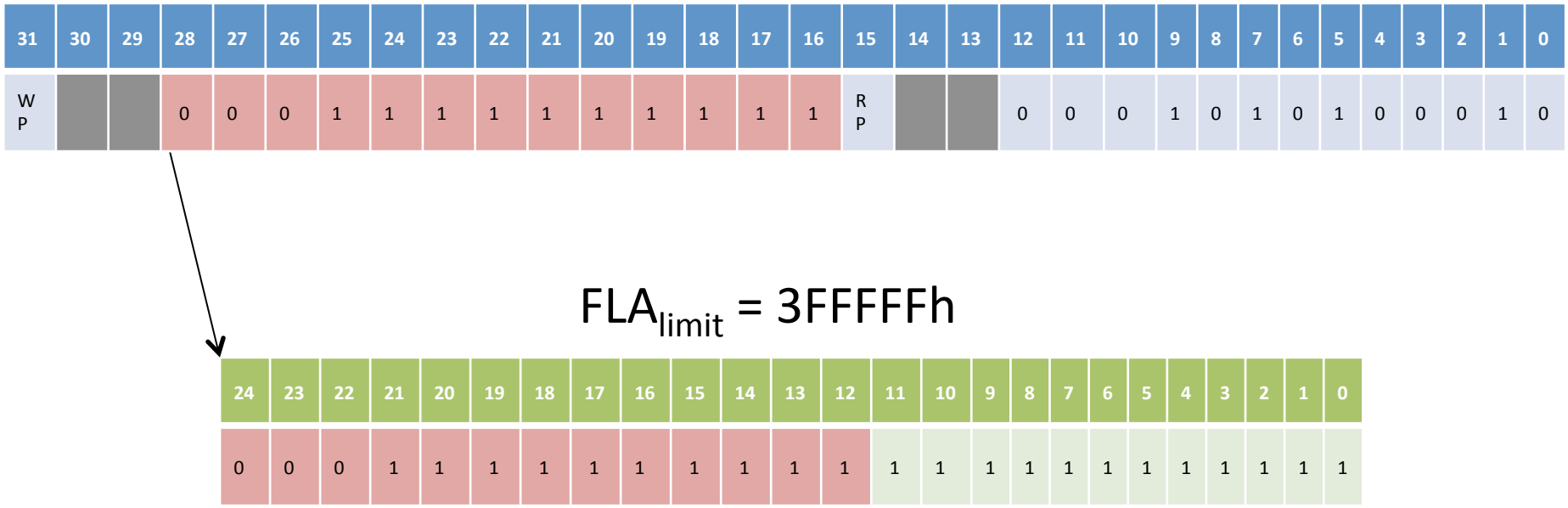
24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0

- To set a PR from a Flash Linear Address Limit:
- PR_{base} = ((page-aligned FLA_{base}) & FFF000) >> 12
- PR_{base} |= 2A2 = 0000002A2h
- To write-protect this range: PR0 | = 80000000h
- To read-protect this range: PR0 |= 00008000h

*We're picking a funny base because the offset at the real BIOS base is normally all 0xFF's so it's harder to illustrate the point. This example will show us a visible boundary whereas the real BIOS base address would not.

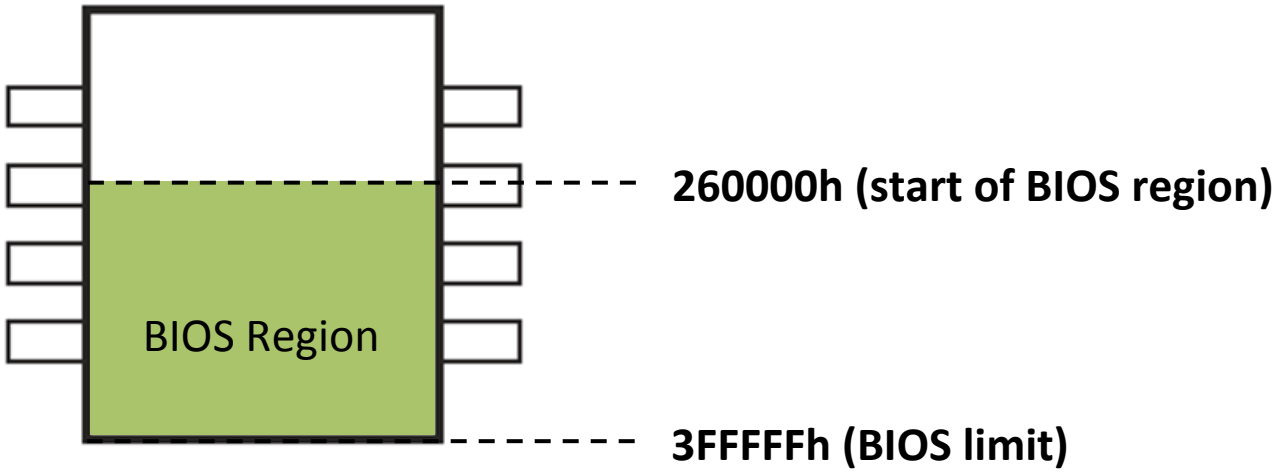
PR Example: 03FF02A2h
Limit 3FFFFFh

Protected Range Limit



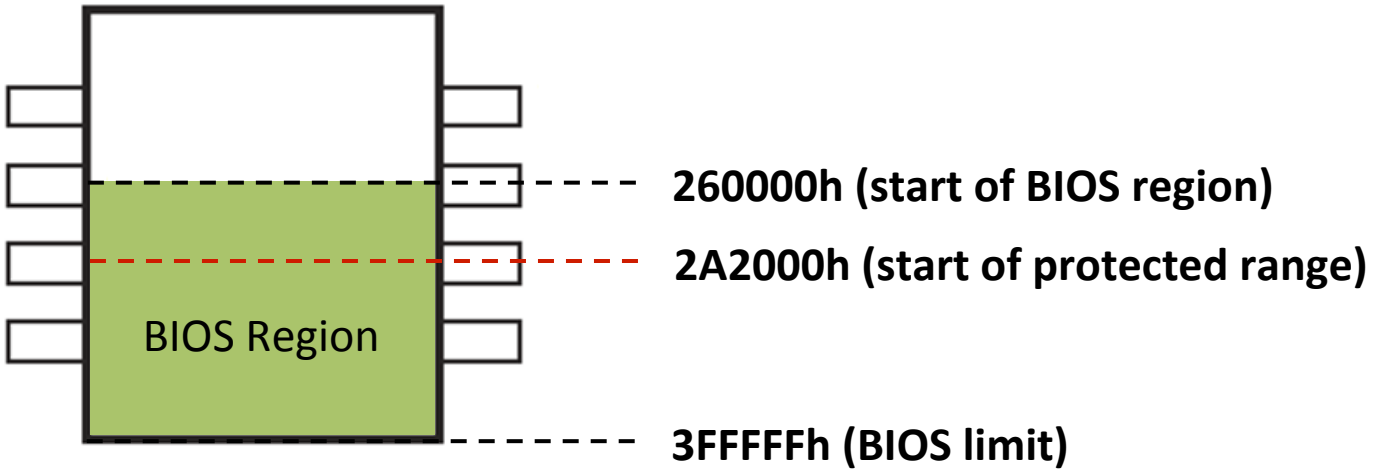
- To set a PR from a Flash Linear Address Limit:
- PR_{limit} = (page-aligned FLA_{limit}) << 16
- PR_{limit} = 3FF000 << 4 = 03FF02A2h
- To write-protect this: PR | = 80000000h
- To read-protect this: PR |= 00008000h

vulnBIOS Example: Protected Range Registers



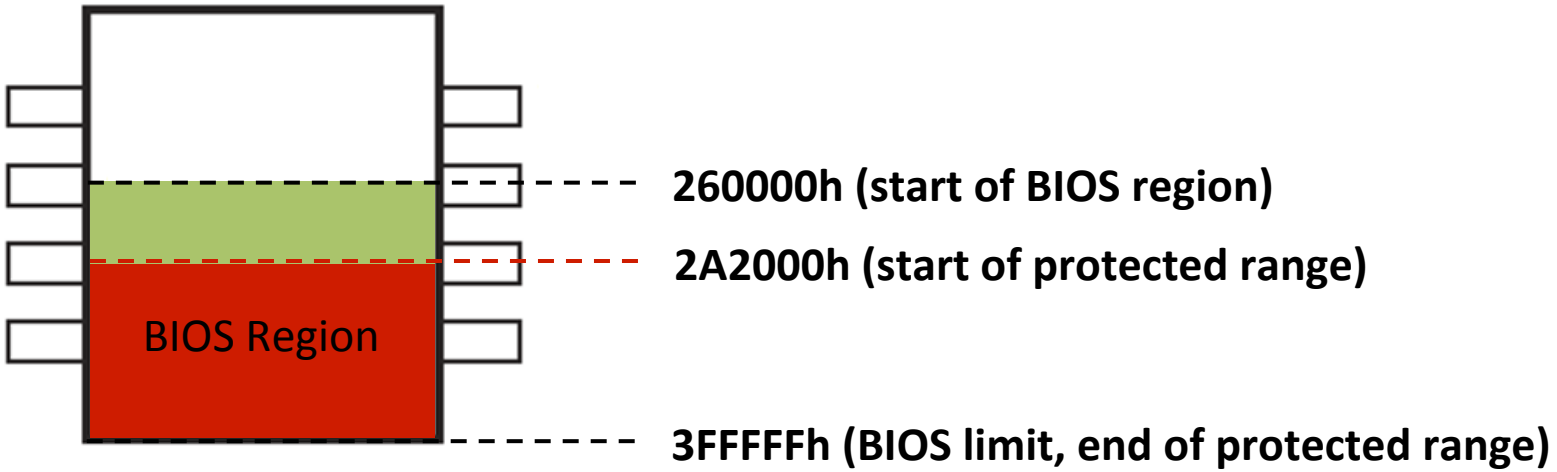
- On our lab E6400, the BIOS region occupies the range 260000 – 3FFFFFh on the physical chip
  - 260000h and 3FFFFFh are Flash Linear Addresses (FLAs)
- The CPU/BIOS (including us using RW-E) always has Read/Write access the BIOS region on flash
  - Per the flash master permission settings

vulnBIOS Example: Protected Range Registers



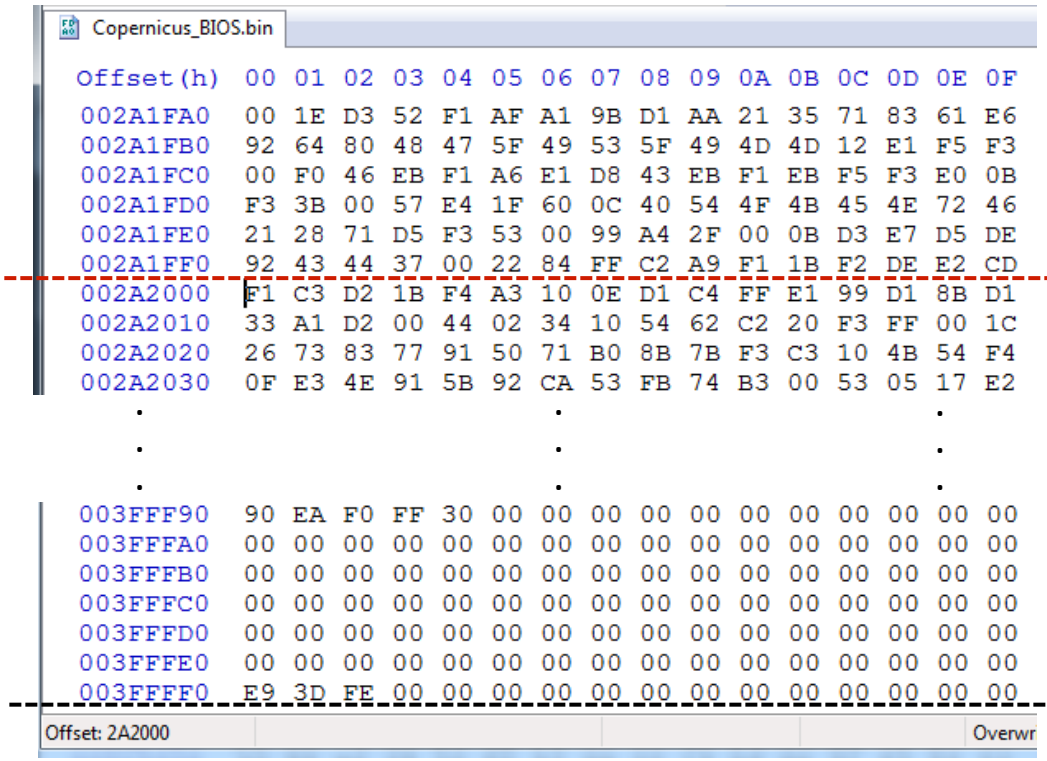
- The previous slides set up a protected range from 2A2000h to 3FFFFFh
- PR = 03FF02A2h (has not yet been read/write protected)
- On our lab machines, this covers a portion of our BIOS region

vulnBIOS Example: Protected Range Registers



- Let's first verify we can read this by viewing the BIOS dump from Copernicus
- PR = 03FF02A2h (has not yet been read/write protected)
- On our lab machines, this covers a portion of our BIOS region

vulnBIOS example: Protected Range Registers

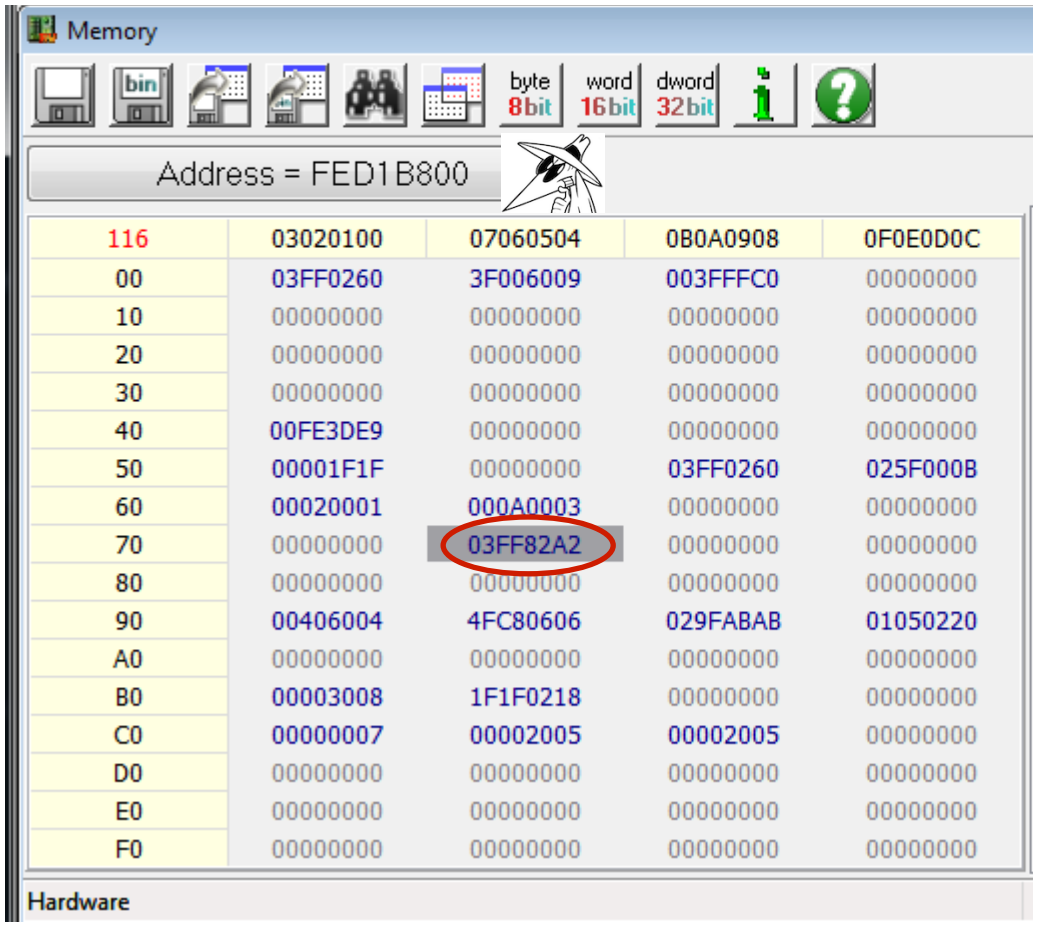


2A2000h (PR Start)

3FFFFFh (PR End (BIOS Limit))

- First let's establish that we have permission to read the BIOS region
- Run Copernicus and open the .bin file with your favorite hex editor (HxD is a good one for Windows)
- Observe binary range 2A2000-3FFFFFh
- Looks like BIOS to me!

vulnBIOS example: Protected Range Registers

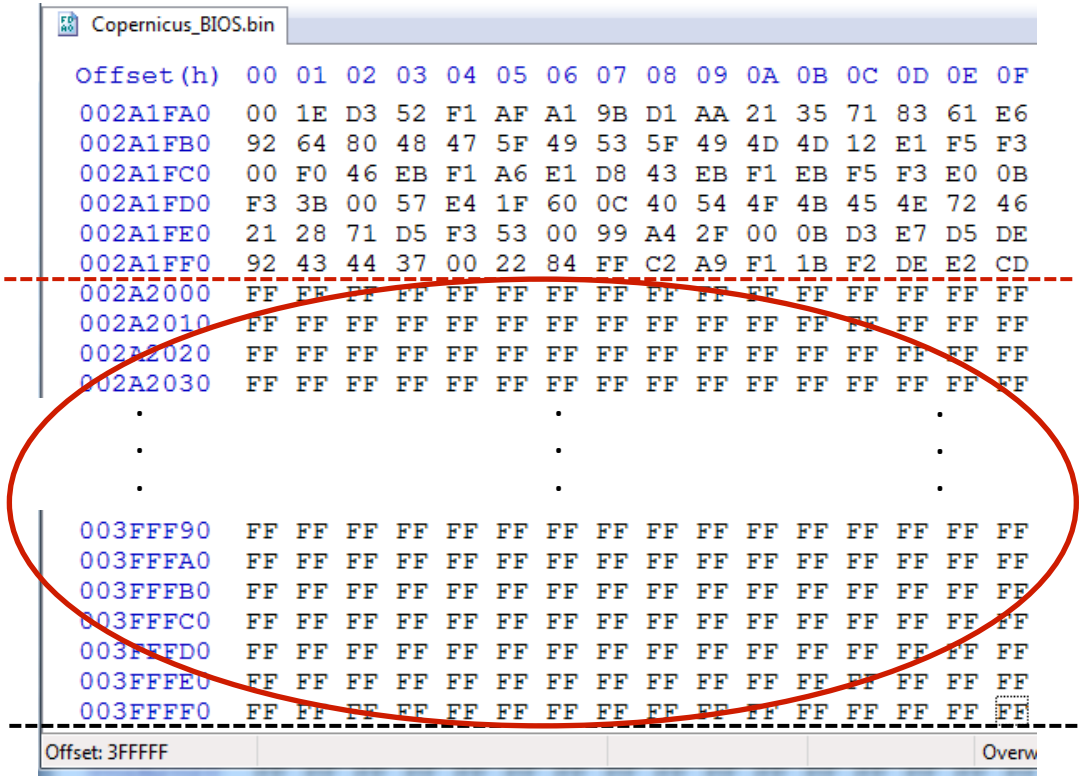


116	03020100	07060504	0B0A0908	0F0E0D0C
00	03FF0260	3F006009	003FFFC0	00000000
10	00000000	00000000	00000000	00000000
20	00000000	00000000	00000000	00000000
30	00000000	00000000	00000000	00000000
40	00FE3DE9	00000000	00000000	00000000
50	00001F1F	00000000	03FF0260	025F000B
60	00020001	000A0003	00000000	00000000
70	00000000	03FF82A2	00000000	00000000
80	00000000	00000000	00000000	00000000
90	00406004	4FC80606	029FABAB	01050220
A0	00000000	00000000	00000000	00000000
B0	00003008	1F1F0218	00000000	00000000
C0	00000007	00002005	00002005	00000000
D0	00000000	00000000	00000000	00000000
E0	00000000	00000000	00000000	00000000
F0	00000000	00000000	00000000	00000000

- To Write-protect a range:
- PR | = 80000000h
- To Read-protect a range:
- PR |= 00008000h

- For this example let's disable reads to this range:
- Set PR0 (at [illegible] + 74h) to 03FF82A2h

vulnBIOS example: Protected Range Registers



2A2000h (PR Start)

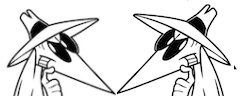
3FFFFFh (PR End (BIOS Limit))

- Now re-run Copernicus and view the BIOS binary file in a hex editor
- As you can see the Protected Range registers override the Flash Master permissions
- This is the BIOS region to which the CPU/BIOS Master otherwise “always” has permission to read and write

PR Summary:

- Implementing Protected Ranges is an important strategy for locking down a system BIOS
- Without PR's, any bypass of SMM's global write-protection means an attacker is automatically able to modify the BIOS
- Protected Range register enforcement:
  - Overrides the Flash Master permissions
  - Prevents Reads/Writes directly from the flash, *even when the processor is running in SMM*
    - We did not demonstrate this, but it is true
- Because of this, BIOS updates (or updates to protected ranges) must be performed before the Protected Ranges are configured by the BIOS
- Only the vendor can reliably configure PR's since new UEFI BIOSes had a region of naturally changing content

FLOCKDN Register

- When we were been able to easily modify (and re-modify) registers in the SPI configuration registers that are designed to protect the system
- For example, we can configure and modify the protected range registers to suit the needs of a lab
- But if we can change them, so can any other app capable of mapping 
- Intel provides the FLOCKDN register to solve this problem

FLOCKDN

HSFS—Hardware Sequencing Flash Status Register (SPI Memory Mapped Configuration Registers)

Memory Address: + 04h
Default Value: 0000h
Attribute: RO, R/WC, R/W
Size: 16 bits

Bit	Description
15	**Flash Configuration Lock-Down (FLOCKDN)** — R/W/L. When set to 1, those Flash Program Registers that are locked down by this FLOCKDN bit cannot be written. Once set to 1, this bit can only be cleared by a hardware reset due to a global reset or host partition reset in an Intel ME enabled system.

- FLOCKDN, when asserted, prevents certain configuration registers/bits in the SPI BAR from being changed
- Once asserted, FLOCKDN cannot be reset to 0 until a reset
  - Or can it? :) *Snorlax & Darth Venamis* on Day 5!
- Although hardware-sequencing is available only in descriptor mode, the FLOCKDN bit still provides register lock-down protection when the flash is operating in non-descriptor mode
  - Called SPI or Flash Configuration Lock-Down bit

FLOCKDN Affected Registers

(see *your* manual, but at the time of original class generation…)

1. Flash Regions Access Permissions Register ()
   - bits 31:24 (BMWAG) and bits 23:16 (BMRAG)
2. Protected Range (PR) registers 0 to 4
   - entire register is locked
3. Software Sequencing Flash Control Register (SSFC)
   - bits 18:16
   - Configure SPI Cycle Frequency (20 MHz, 33 MHz, or 50 MHz [PCH only])
4. Prefix Opcode Configuration Register (PREOP)
   - entire register is locked
5. Opcode Type Configuration Registers (OPTYPE)
   - Entire register is locked
6. Opcode Menu Configuration Register (OPMENU)
   - Entire register is locked

SPI Lockdown Summary 1

- Locking down the SPI Flash is a little more complicated than locking down SMM
- For the most part, only the vendor can do this, but you can verify and try to implement some yourself

- Verify that BIOS_CNTL.BLE is set
  - Oh wait…we're going to talk about something in a sec that completely bypasses BLE :)
  - If it's not set, you can assert it yourself but that doesn't mean there is SMI handler code present that will de-assert bit 0
- Verify that SMM is protecting the BIOS from writes by asserting bit 0 and ensuring that it is reset
- If supported, ensure that BIOS_CNTL.SMM_BWP is asserted so that the BIOS can only be written to when the processor is in SMM
  - You can set this yourself. The only drawback being that you may not be able to update the BIOS, depending on how the vendor implemented updates

SPI Lockdown Summary 2

- Verify that Protected Range registers are being used
  - You could also set these yourself but it will be a trial and error exercise since you won't know what parts of the BIOS flash will be used to store variables (UEFI definitely and some Legacy)
- Set FLOCKDN to ensure the above registers can't be changed
- The above changes you could play with won't permanently hurt your system if they lock it up – they will all reset back to their original values on startup

- Verify that the Flash Master permissions are set and that the Flash Descriptor region cannot itself be written to
  - You have no control over this unless it is writeable, in which case the most you should do is make the FD un-writeable
  - Messing with this one could brick your system "permanently"

SPI Summary

- Locking down the SPI flash memory is the first line of defense against an attacker
- It is complicated and hard for vendors to get right
- It gets a little more complex in UEFI where the SPI flash is specifically used as a file system for storing system variables
  - Can't just set a single PR to write-protect the whole BIOS region
- Remember:
- The BIOS boots from the flash and is responsible for configuring all of the settings we have been discussing so far in the class
- Letting an attacker modify the BIOS means game over
- It's not easy, but it's not that hard either for an attacker to modify your BIOS flash

SPI Summary

- All the settings in this section apply to both x86 and x64 architecture
- All the settings in this section apply to both legacy BIOS and UEFI BIOS
- All the settings in this section apply to systems running legacy MCH/ICH chipsets and the new PCH chipsets
  - Except where otherwise noted (SMM_BWP)