# Advanced x86:
# BIOS and System Management Mode Internals
## *PCI {Option/Expansion} ROMs*

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).

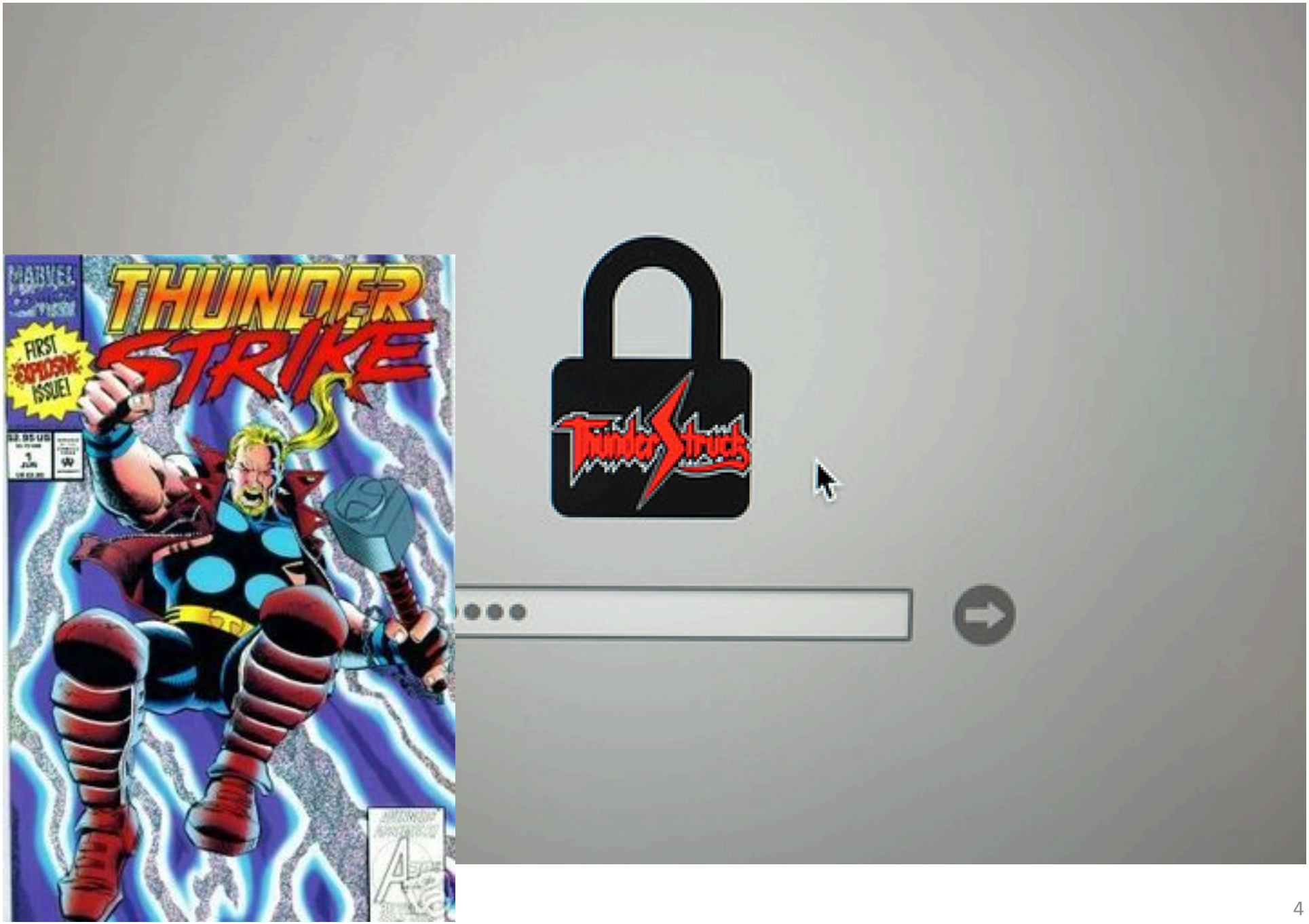
**Share Alike** — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

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”

# PERSISTENCE
## PCI DEVICE EXPANSION ROMS

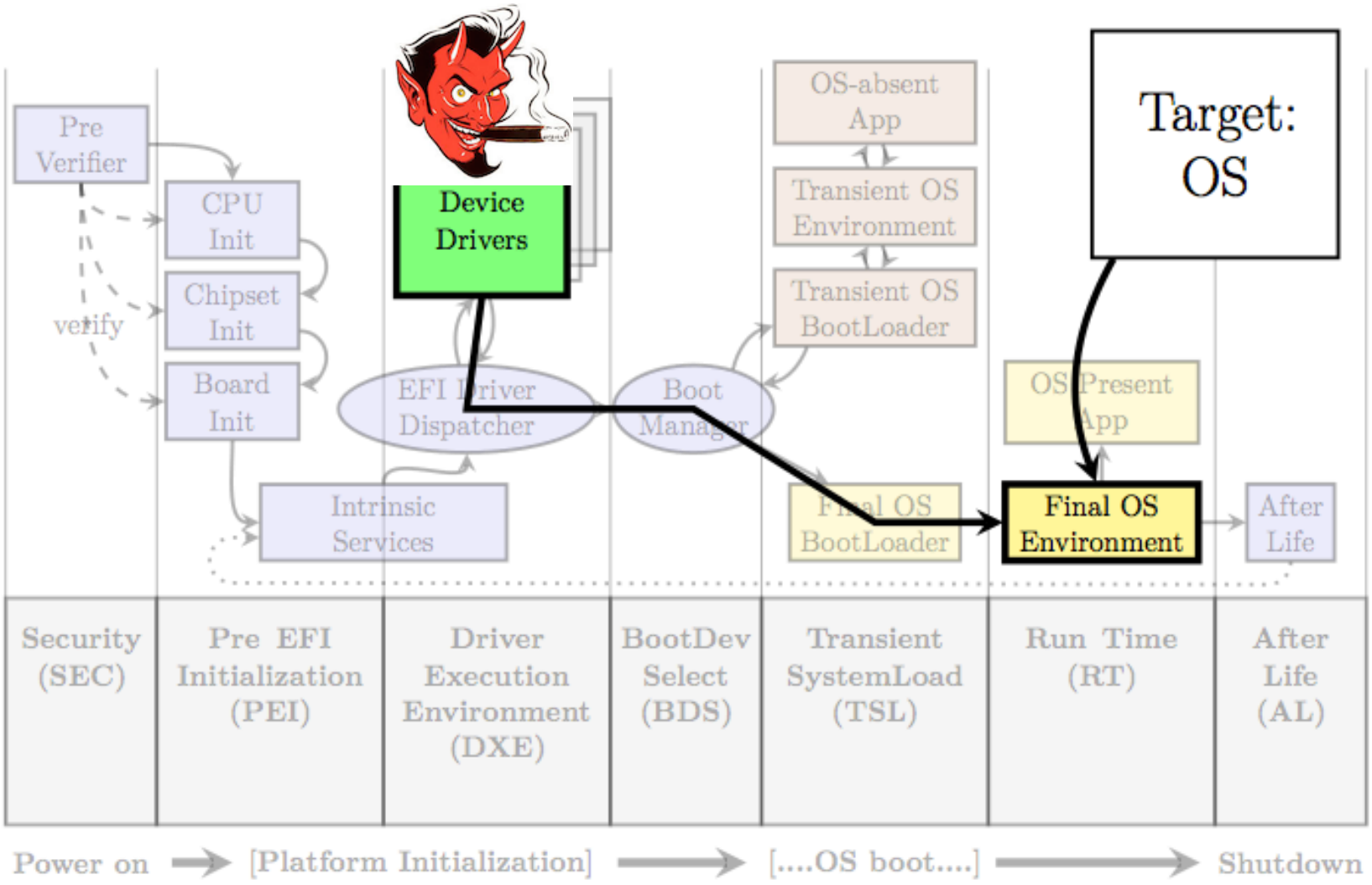
- Hardware-specific
- Graphics cards in iMacs have them
  - MacBook Pros too
  - My old test MacBook - no dice
  - VMware's ethernet interfaces do - hurr (good for testing)
- Can write to them from the OS
  - Thanks, iMacGraphicsFWUpdate.pkg!
  - Probably with flashrom
- Pretty awesome. 7/10.

https://trmm.net/Thunderstrike_31c3



Pierre Chifflier, UEFI and PCI Bootkits, PacSec 2013 [34]

# Scenario

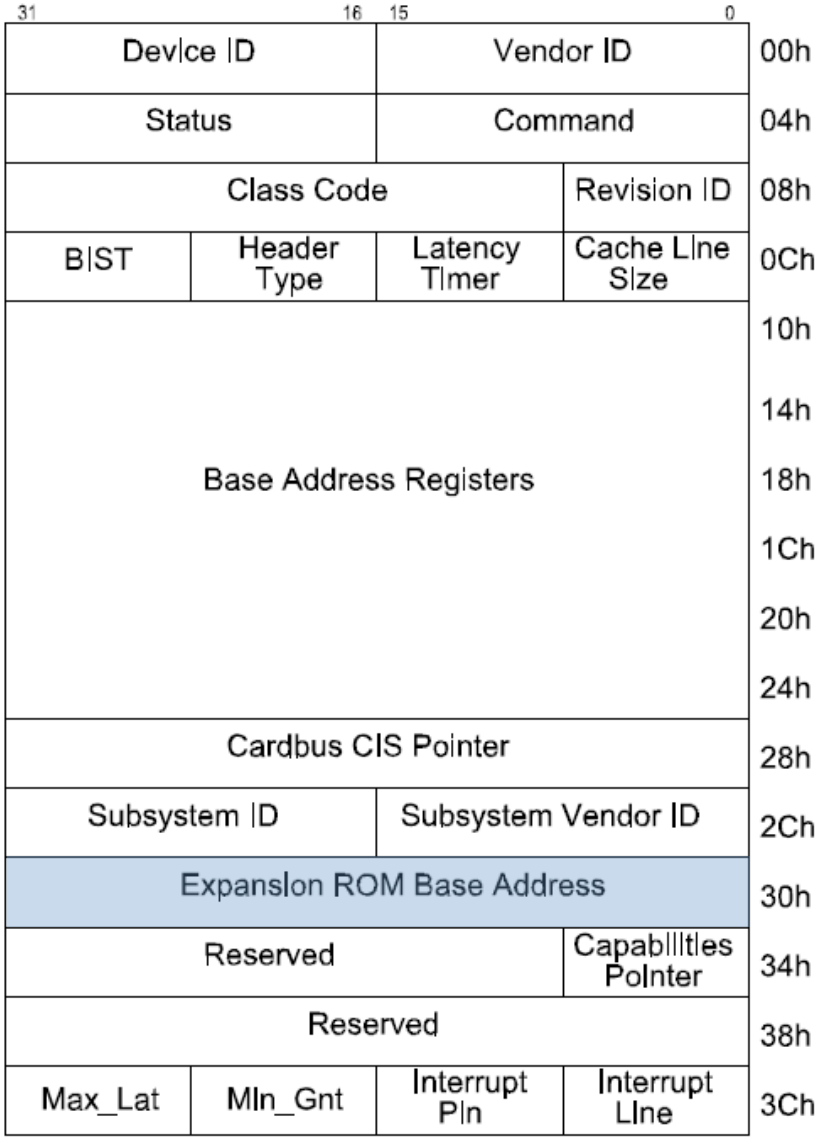



2σ
TWO SIGMA
LEGBACORE
Thunderstrike 2:
Sith Strike
Trammell Hudson – Two Sigma
Xeno Kovah, Corey Kallenberg – LebgaCore

# PCI/PCIe Expansion ROMs (XROMs) aka Option ROMs (OROMs)

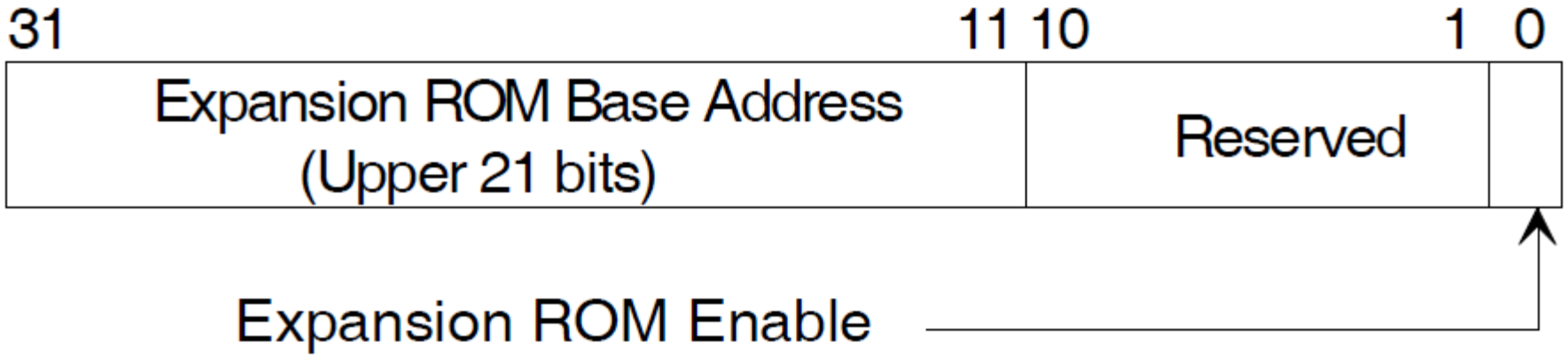
- A PCI/PCIe Expansion ROM is *x86 native executable code* located on a PCI device
  - Can technically have multiple architectures' native code on it, so that the device can load just as well on a PPC device as an x86 one.
- Not every device will have one
  - Graphics cards, network cards will likely have one
  - A device can have multiple XROMs (for multiple architectures)
- Benign or otherwise this code gets executed by the CPU/ BIOS during the boot process
- They are handled the same on PCI Express as they are in PCI
- They are configured via a separate BAR called the Expansion ROM Base Address Register

# Expansion ROMs



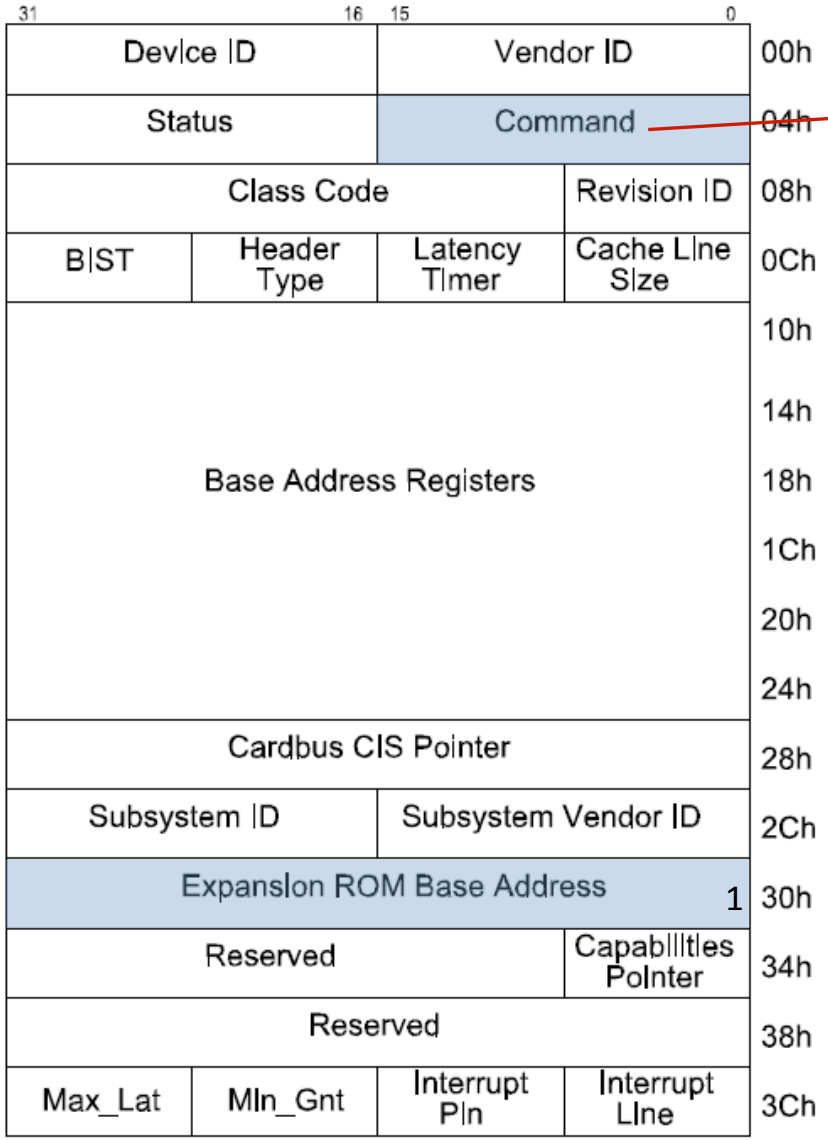

- XROMs have their own BAR called the Expansion ROM Base Address Register
  - On general type PCI devices it's located at offset 30h
  - On bridge type devices it's at 38h
- BIOS initializes the XROM BAR like the other BARs, but hands off execution control to the code it points to
- XROMs are copied to memory before being executed
  - On legacy systems they are copied to C0000 to DFFFFh range
- The XROM BAR operates similarly to the other BARs but the interpretation of the field's bits is slightly different

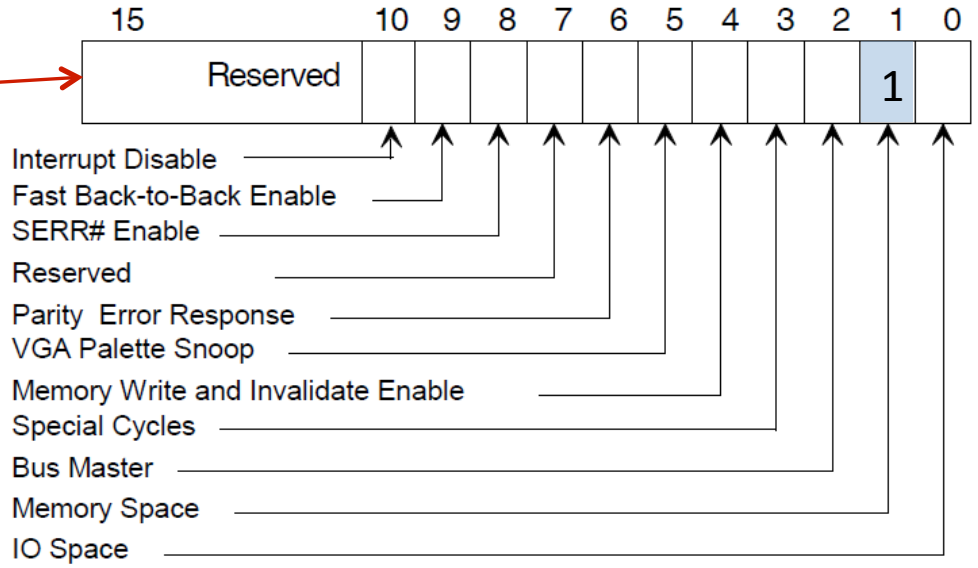
# Expansion ROM Base Address Register




- The LSB determines whether accesses to the Expansion ROM are permitted. When asserted to 1, they are permitted
- Even when a device has an Expansion ROM, its BAR may still be 0 (meaning access to it is not permitted)
- Like the PCI BARs, the Expansion ROM BAR is also R/W

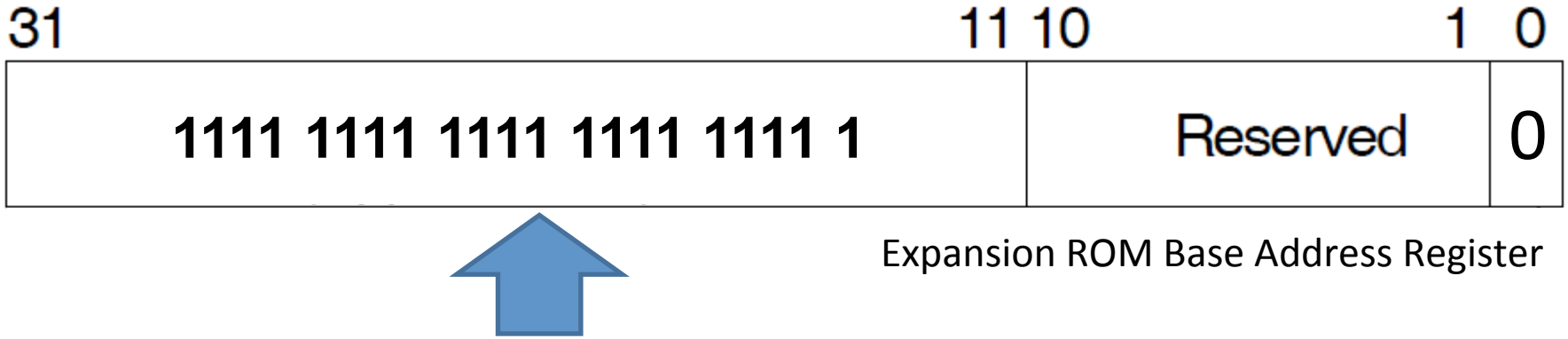
# Command Register and Address Space Access



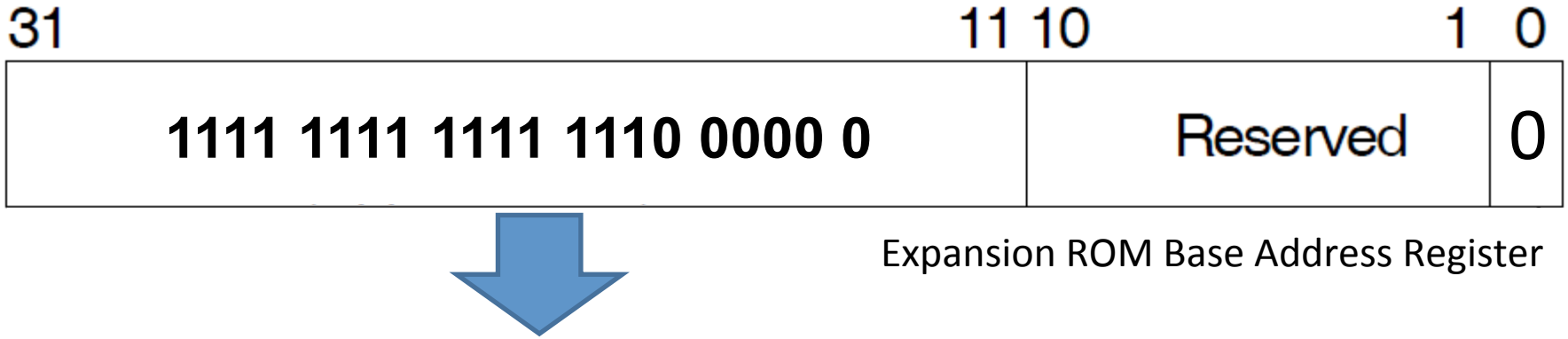

- An expansion ROM will only respond to accesses if the Expansion ROM Enable bit *and* the memory space bit in the Command Register are both set

# How CPU/BIOS Discovers XROMs



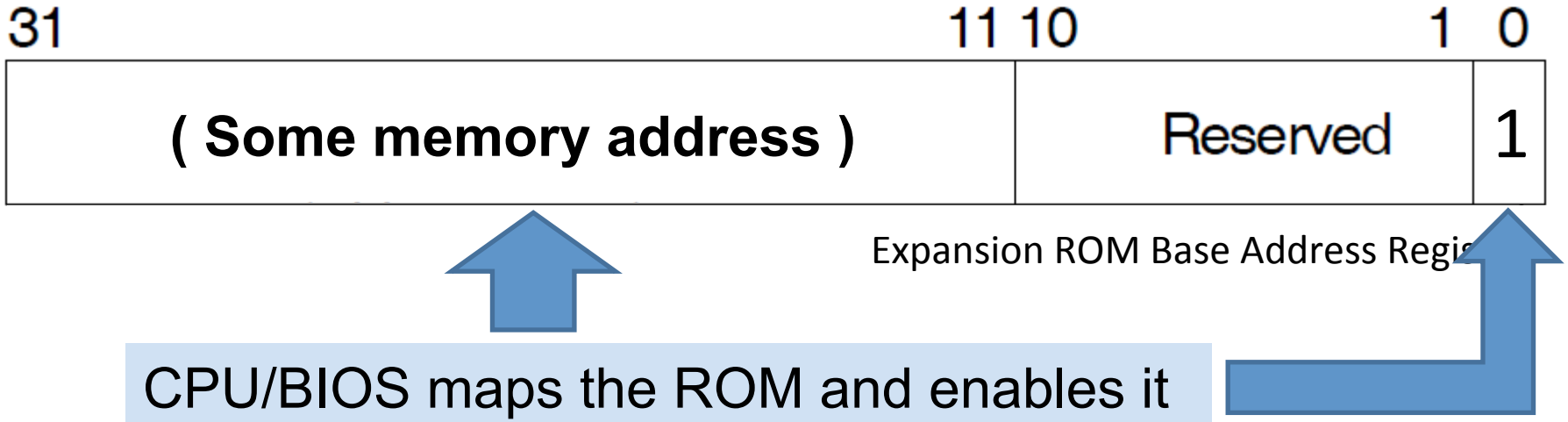

- To determine whether the device has implemented an Expansion ROM base:
- All 1's are written to the top 21 bits (31:11) of the Expansion ROM BAR
- If the device returns anything other than 0, then it has implemented an Expansion ROM

# How CPU/BIOS Discovers XROMs



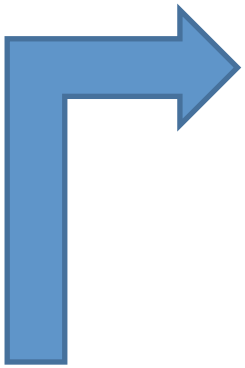

- The return address indicates both the size of the ROM and the memory alignment (mask) required by the ROM:
- Per the above example:
- Size = ~FFFE_0000 +1 = 2_0000h bytes
- ROM must be mapped to a 128KB-aligned memory address
  - So addresses like XXX00000, XXX20000, XXX40000, etc

# How CPU/BIOS Discovers XROMs




- Next the CPU/BIOS maps the ROM to an unused portion of memory
- Then it sets the enable bit so that the ROM is now accessible at the address defined by the BIOS

# How CPU/BIOS Discovers XROMs



| Offset | Length | Value | Description |
|---|---|---|---|
| 0h | 1 | 55h | ROM Signature byte 1 |
| 1h | 1 | AAh | ROM Signature byte 2 |
| 2h | 1 | xx | Initialization Size - size of the code in units of 512 bytes |
| 3h | 3 | xx | Entry point for INIT function. POST does a FAR CALL to this location. |
| 6h-17h | 12h | xx | Reserved (application unique data) |
| 18h-19h | 2 | xx | Pointer to PCI Data Structure |

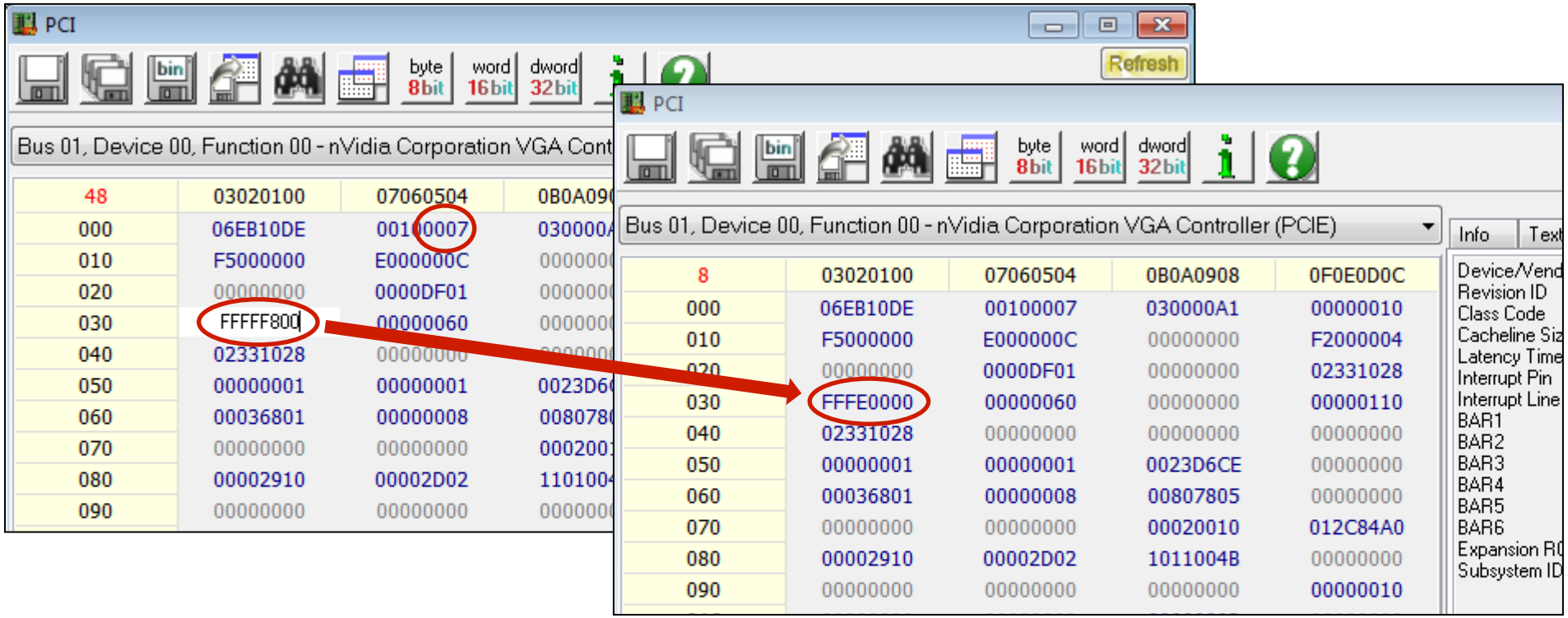
CPU/BIOS checks memory for Option ROM structure

- If anything other than the “AA55” signature is present, there is actually no Option ROM provided by the device, despite the fact that it returns a mask as if there were
  - I have some ice cream. Want a lick? Psych!
- There may still be an option ROM, however, some companies implement them in non-standard ways

# CPU/BIOS Expansion ROM Discovery

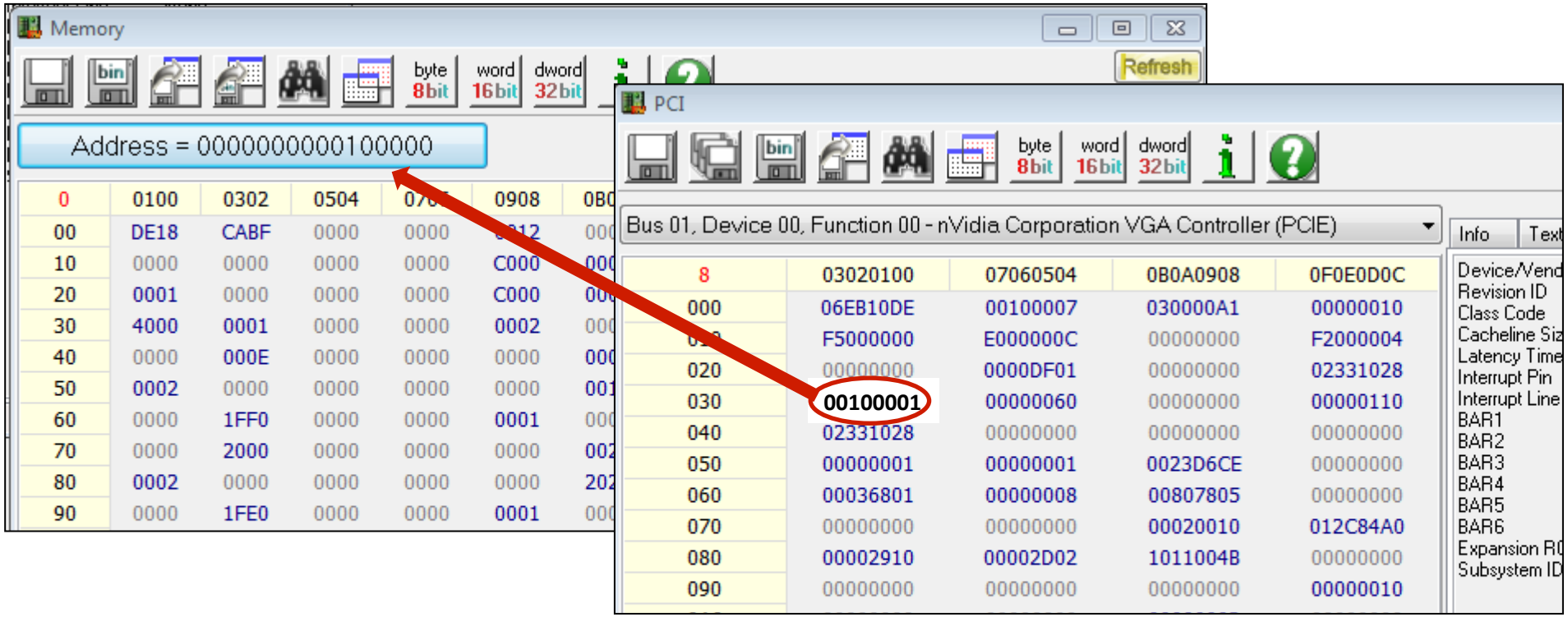
- A PCI device can share a decoder between the Expansion ROM BAR and other BARs
- For example:
- Some vendors mirror their Expansion ROMs at BAR[n] or at an offset from BAR[n]
  - NVidia sometimes puts them at BAR[0] + 30_0000h (per the developers of Flashrom )
  - http://flashrom.org/Flashrom
- It is possible that there simply is no Expansion ROM present on the device
  - *Could be located in a compressed module in the BIOS binary*

# Expansion ROM Discovery: User Example (Same as BIOS)



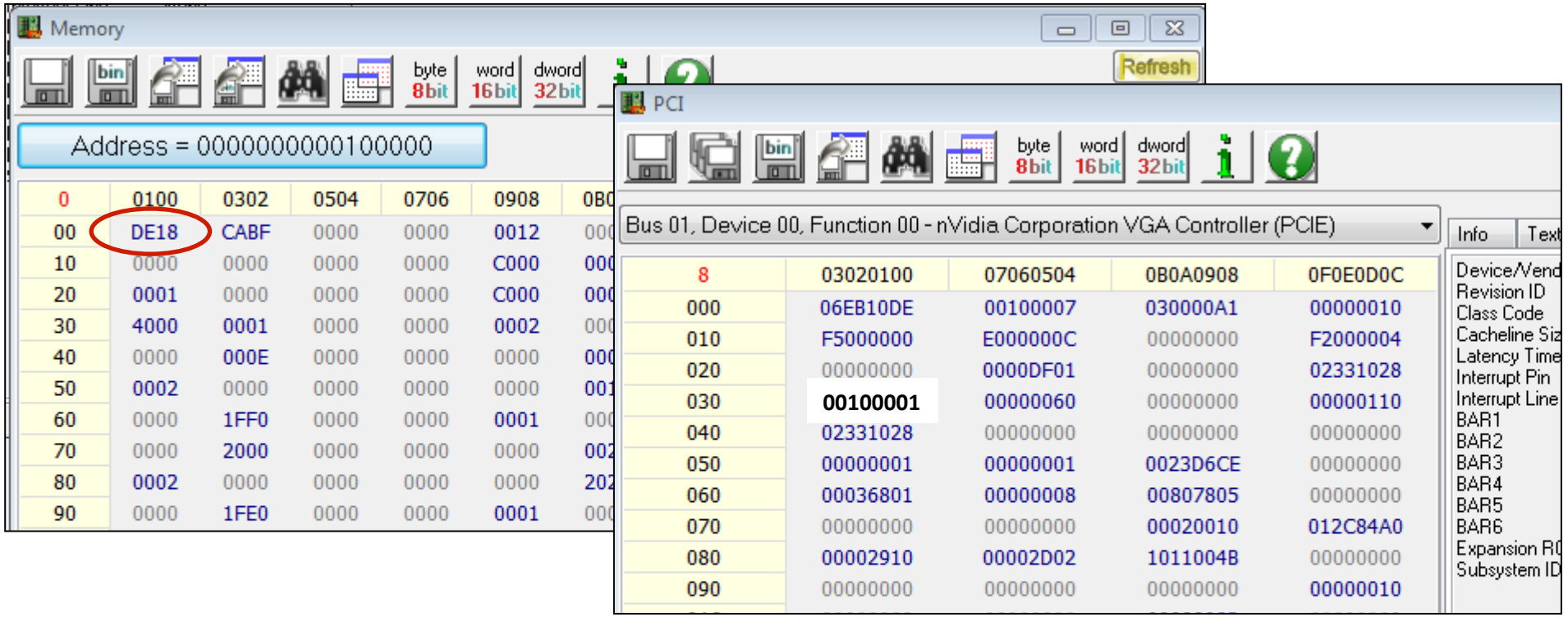

- This example pertains to the nVidia VGA card on the E6400 laptop
- Verify that the memory-enable space bit 1 in the command register (offset 04h) is asserted
- Writing FFFF_F800h to offset 30h returns FFFE_0000h indicating that an Expansion ROM [might be] present
  - Bit 17 is the LSB, which indicates a 128KB ROM
  - Size = ~FFFE_0000 + 1 = 2_0000h bytes

# Expansion ROM Discovery: User Example (Same as BIOS)




- We (or the BIOS) should be able to choose a memory address for the ROM to be mapped to
- Address must meet alignment requirements
- Address must provide enough room for the XROM
- Must enable the XROM decoding (assert bit 0, enable)

# Expansion ROM Discovery: User Example (Same as BIOS)




- If there is anything other than the “AA55” XROM signature, then there is actually no option ROM present
- As it turns out, in this case, there is no option ROM located on the device
- This option ROM is located on the BIOS flash as a compressed module

# Expansion ROM Hacking

- Hacking an Expansion ROM typically requires reflashing the firmware on the device
  - Often the “RO” in “ROM” is a misnomer
  - Although in the case we just saw, modifying the BIOS itself could permit an attacker to insert a malicious XROM
- If a vendor offers a utility to update the flash then you know the flash is writeable
- Good reference on XROM hacking:
- http://resources.infosecinstitute.com/pci-expansion-rom/
- It’s important for Option ROMs to be measured (measured boot) before being executed

# Secure Boot

- Systems that support UEFI/Windows 8 Secure Boot require XROMs to be signed before it will execute them
  - Assuming you didn't turn off SecureBoot
- Apple systems don't support SecureBoot, therefore what worked in 2012 still works today
  - The fact that systems load XROMs off external peripherals like the Thunderbolt Ethernet adapter make it just that much easier to attack Macs this way

# References


- https://sites.google.com/site/pinczakko/building-a-kernel-in-pci-expansion-rom (Darmawan Salihun)
- http://www.blackhat.com/presentations/bh-dc-07/Heasman/Paper/bh-dc-07-Heasman-WP.pdf (John Heasman)
- http://pacsec.jp/psj13/psj2013-day2_Pierre_pacsec-uefi-pci.pdf (Pierre Chifflier)
- http://ho.ax/downloads/De_Mysteriis_Dom_Jobsivs_Black_Hat_Slides.pdf (Snare)
- https://trmm.net/Thunderstrike (Trammel Hudson)
- http://legbacore.com/Research_files/ts2-blackhat.pdf (Trammel Hudson, Xeno Kovah, Corey Kallenberg)