

# Firmware Rootkits: The Threat to the Enterprise



John Heasman, Director of Research

**NGS** Consulting

# Agenda

- Recap of ACPI BIOS rootkit and limitations
- Brief overview of the PCI Bus
- Abusing expansion ROMs
- Abusing PXE
- Detection, Prevention and the TPM
- Summary and conclusions

# Rootkit Persistence

- Rootkits on disk subject to “cross-view detection”
- Current rootkit detection tools consider only disk
- But many devices have firmware...
- Objectives for rootkit writer:
  - Persist rootkit in firmware
  - Automatic load before/during OS boot
  - Bootstrap component on disk is cheating!

# Abusing ACPI

- BIOS holds tables containing AML instructions
- ACPI device driver contains AML interpreter
- AML instruction set allows us to modify system memory
- Re-flash BIOS to contain patched ACPI tables
- AML methods now deploy rootkit from BIOS

# Limitations of ACPI Rootkits

- Must be able to update system BIOS
  - Signed updates prevent attack (Secure Flash)
- OS must have ACPI device driver
  - Stop it loading for cross-view detection
- OS must not sandbox AML interpreter
  - Prevent mapping of kernel address space

# Persistence via PCI

# Open Questions

Consider each and every machine on your network:

- What PCI devices are present?
- Where were they sourced from?
- Which of these have flashable firmware?
- What firmware is currently on each device?
- Can you trust the integrity of the firmware?

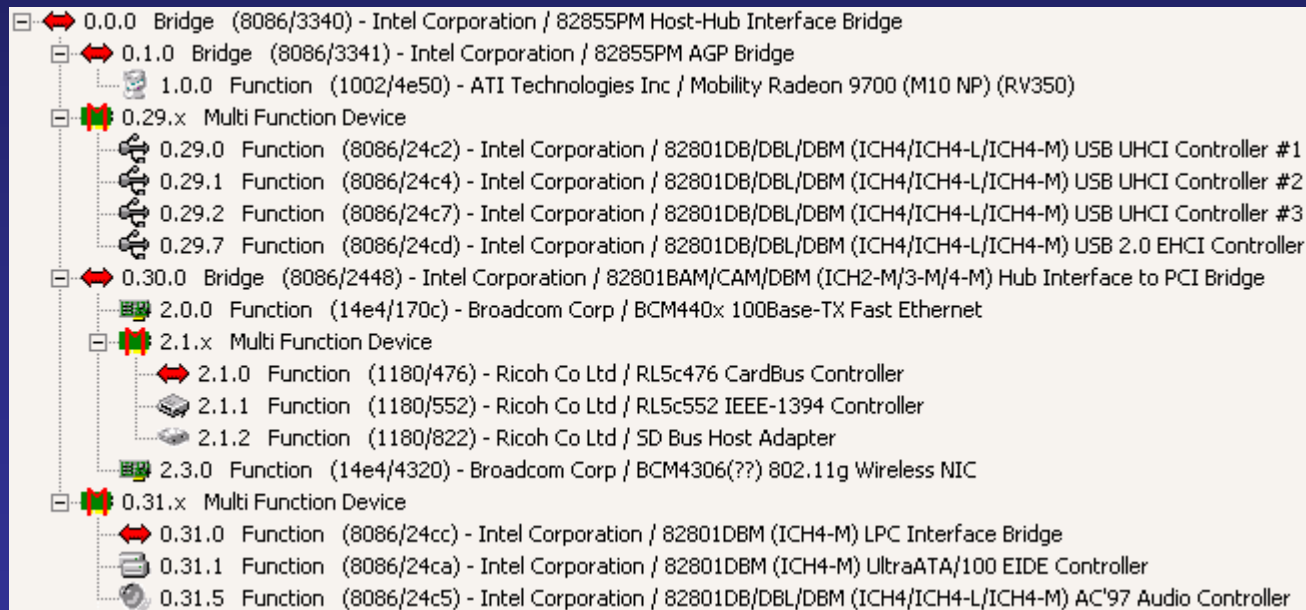
# Introduction to the PCI Bus

- Bus for attaching peripherals to motherboard
- Developed by Intel circa 1990
- OS queries all PCI buses at start up
  - Find out what devices are present
  - Find out what resources each needs
- PCI Configuration space holds device type information
  - Helps OS choose device drivers
  - Also contains human readable device identification



# A Typical PCI Bus

- Bridges connect multiple buses together
- Devices can have multiple “functions”



# PCI Expansion ROMs (1)

- ROM on PCI card holding initialisation code
- Can be for any platform but typically holds x86 code
- Copied to RAM and executed by system BIOS
- Stored in EPROM or EEPROM
- Example: EEPROM on your PCIe graphics card:
  - Hooks int 10h in real mode IVT
  - Implements VGA/VBE BIOS functions

# PCI Expansion ROMs (2)

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

# Subverting the Kernel

- Modify an expansion ROM to subvert the NT kernel
- We can do this via hooking interrupts
  - eEye's BootRoot hooks int 13h (disk)
- Alternative technique: hook int 10h (video)
  - Called by system BIOS
  - And NTLDR...
  - And NTOSKRNL...
  - But when and how do we execute our payload?

# Recap of x86 Operation Modes

- Real Mode
  - 20 Bit segmented memory addressing
  - Direct software access to BIOS routines
  - No concept of memory protection nor multitasking (at least at hardware level)
- Protected Mode (p-mode)
  - Memory protection
  - Hardware support for virtual memory task switching
  - Paging system
  - Four privilege levels (rings 0 – 3)

# Virtual 8086 Mode

- Also called virtual real mode (or VM86)
- Allows the execution of real mode applications that "violate the rules" under the control of a p-mode OS
- Segmentation mechanism works like real mode
  - But paging mechanism is still active
  - So memory protection is still applicable

# Ke386CallBios

- During boot, Windows calls int 10h from Po8263de:nt

# Executing the Payload

- Return from this int 10h is a special case
  - Detected by specific registers values in VDM TIB
  - ESI of return context points to pmode stack frame
  - VDM TIB is at fixed location (0x12000 virtual)
- We can't modify the register values directly
  - The fault handler must recognise special case
  - But we can modify the stack frame pointer!
- We can set up a fake stack frame to return to us
  - And execute 32-bit pmode code in ring 0 😊



# Zero Cost Development with Bochs

- Bochs is an Open Source x86 emulator
  - Contains an integrated debugger
  - Can also be used with GDB
- We can create and debug our own expansion ROMs
  - And breakpoint execution of real mode interrupts
  - VGABIOS is an LGPL'd VGA BIOS

# Advanced Payload Features

- Remove hardcoded addresses:
  - Determine kernel version from KPCR
  - Map and execute code directly from ROM
  - Or decompress to allocated memory
- Provide update mechanism:
  - Re-flash update on to host card
  - Use TDI or NDIS to connect to rootkit controller
  - But, this could be caught by a personal firewall
- If we could update pre-boot, HIDS/HIPS can't catch us

# Abusing PXE

# Intel's Preboot Execution Environment

- Expansion ROM(s) on NIC implement DHCP, TFTP
- Used as Initial Program Load (IPL)
- Used in diskless workstation, and to ghost machine
- Implemented as a monolithic ROM (older NICs)
- Or multiple modular ROMs

# PXE Components

- Universal Network Driver Interface (UNDI) API
  - Lowest level PXE API for sending/receiving frames
  - Network card specific
- Pre-boot API
  - Initialises the UNDI ROM
  - Starts execution of base code
- TFTP and UDP API
- Base Code
  - The application that uses these APIs

# Abusing PXE

- BC Loader Hook:
  - Check for magic return address
  - If detected, execute alternate base code
- Init of any ROM:
  - Hook int 19h to get control at boot
  - At boot, locate and call UNDI IPL
  - Restore original int 19h to continue normal boot
- Alternate base code:
  - Contact rootkit controller
  - Download update

# Etherboot

- Open source PXE ROM creation tool
- Supports many NICs and protocols:
  - ARP
  - TCP (HTTP)
  - UDP (DHCP, NFS, DNS, TFTP)
- Easy, flexible alternative to patching existing ROMs
- But also easier to detect (though could be made harder)

# Modifying Etherboot

- Develop a ROM to:
  - Hijack the IPL table to get control at boot
  - Send a UDP heartbeat disguised as DNS
  - Check response for update status
  - Download update by TFTP if present
  - Continue boot as normal
- Trivial improvements:
  - Encrypt and compress update code
  - Schedule updates
  - Combine with covert channel for data extraction



# Implementing PXE on other cards

- “Flaw” in PCI/PNP specification:
  - ROM on any PCI card can implement PXE!
  - Thus NIC does not need to be flashable!
  - Could put it in the system BIOS (if we can flash it)
  - Could split implementation across multiple cards

# Detection & Prevention

# Malicious ROM Detection (1)

- Scan all ROMs in system memory
- Can use /device/PhysicalMemory prior to Win2k3 SP1
  - Though better to do this from kernel anyway
- Disassemble ROM and ask:
  - Is it a known good image?
  - Which interrupts does it hook?
  - Does it contain 32-bit code?
  - Any suspicious strings or addresses?
  - What does it actually do?

# Malicious ROM Detection (2)

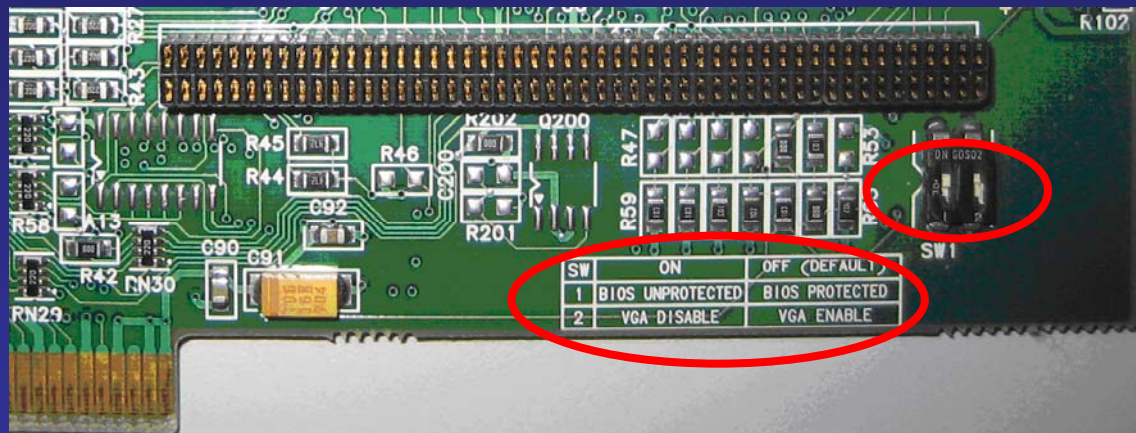
- Often, ROM in memory != ROM on card
- This is deliberate
  - To save space, init code is often discarded
  - Image size, checksum fixed up during init
- We need to analyse the init code as well
  - So we have to dump the ROM off card itself
  - Documented in the PCI spec

# Difficulties in Detection

- Signature checking of ROMs is insufficient
- Analogous to AV/IDS detection of exploit variants
  - What about polymorphism & self-modifying code?
- Can perform limited dynamic analysis using VM
  - What about hardware specific operations?
- Static analysis might be the only way
  - Complex and doesn't scale
- Currently no repository of known good ROMs

# Prevention (1)

- Is hard! Most PCI cards:
  - Do not have write-protect jumpers
  - Do not require signed firmware updates



# Prevention (2)

- Cards are reflashed via I/O to PCI configuration space
- Vendor utilities likely to use:
  - HalGetBusData (Obsolete)
  - IRP\_MJ\_PNP: IRP\_MN\_WRITE\_CONFIG
  - These could be hooked and analysed
- But these just wrap I/O instructions
  - Attacker can do required I/O directly!

# Prevention (3)

- NtSetInformationProcess allows modification of IOPL
  - Have to have SeTcbPrivilege
  - LocalSystem can therefore do (unrestricted) I/O
  - Admin can get LocalSystem...
  
- Worst case scenario:
  - User runs browser as Admin
  - Browser owned by exploit
  - Exploit reflashes graphics card
  - No driver needed!



# Trusted Platform Module (1)

- Is a microcontroller on the motherboard that:
  - Performs crypto functions (RSA, SHA-1, RNG)
  - Can create, protect and manage keys
  - Contains a unique Endorsement Key (an RSA key)
  - Holds platform measurement hashes
- The Secure Startup process builds on the TPM to:
  - Measure each system boot event
  - Store hashes in Platform Configuration Registers
  - Compare against PCRs on subsequent boot ups

# Trusted Platform Module (2)

- Expansion ROMs hashes are stored in a PCR
  - Prevents modification after set up of Secure Startup
- Caveats:
  - This assumes card is not already trojanned!
  - What is a ROM is supposed to look like?
  - How many cards have a “reset to factory default”?
  - Do you trust the factory? 😊

# Summary

- Expansion ROMs typically hold x86 code
  - Executed during POST, before OS boot
  - Subvert OS boot to deploy/bootstrap rootkit
- PXE provides a pre-boot means of using the network
  - Expansion ROMs can use this to update rootkit
  - And provide a pre-boot covert channel
- Detection focuses on analysis of expansion ROM
  - Signature and heuristic detection
- TPM with secure bootstrap prevents this class of attack

# For Further Discussion

- How long before widespread TPM adoption?
  - And secure bootstrap implementations for every OS?
  - 5 years? 10 years? Ever?
- Just whose problem is this and how do they/we fix it?
  - Vendors should add jumper to prevent update?
  - Or “return to factory default” switch?
  - Or hashes for all firmware revisions?
- Will we ever see malware target firmware?
  - Will there ever be sufficient ROI?

# References

## PCI Specification

<http://www.pcisig.com/specifications>

## PXE Specification

<http://download.intel.com/design/archives/wfm/>

## Etherboot

<http://www.etherboot.org>

## Bochs

<http://bochs.sourceforge.net/>



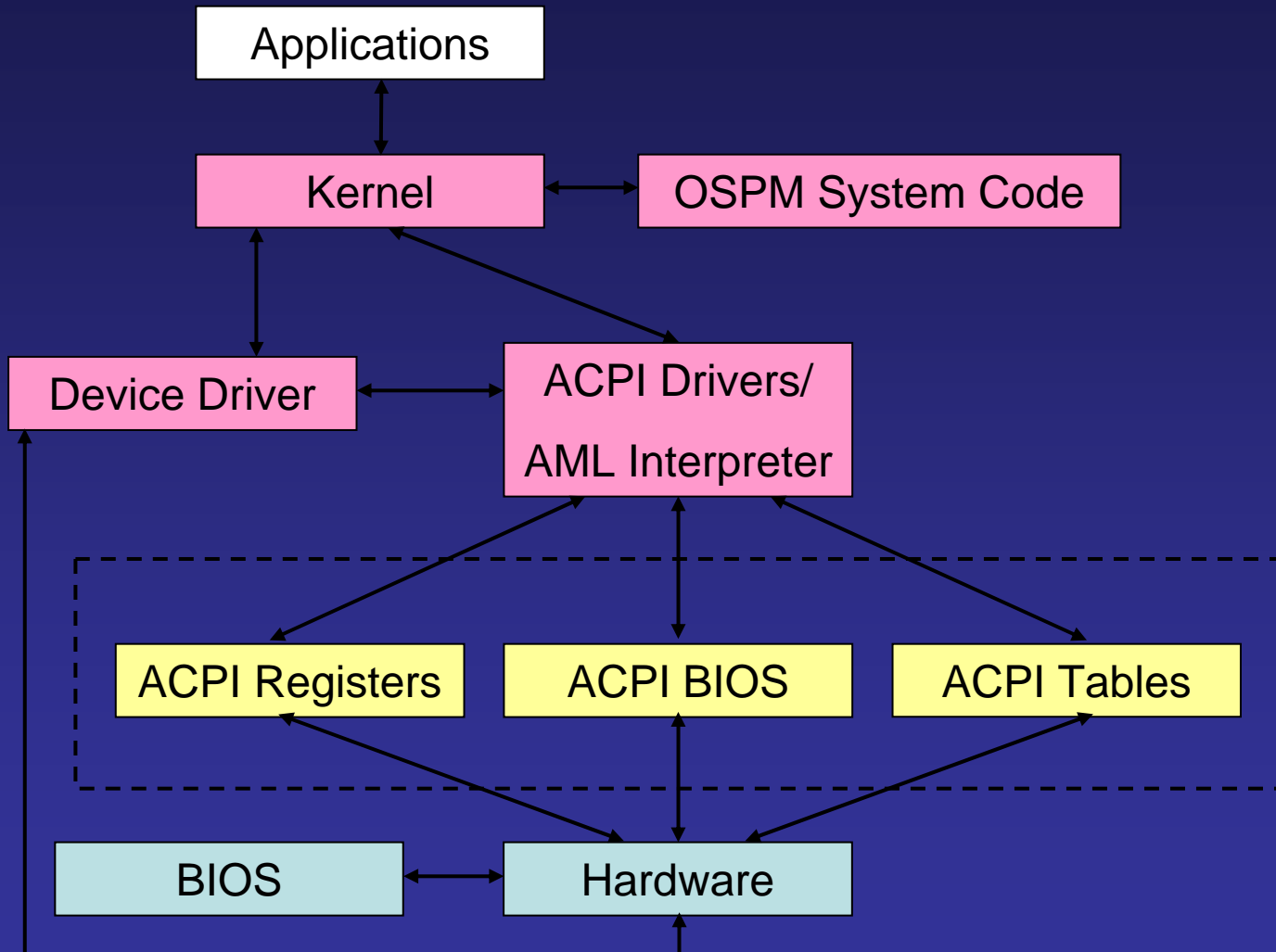
Any Questions?

Thanks!

john at ngssoftware dot com

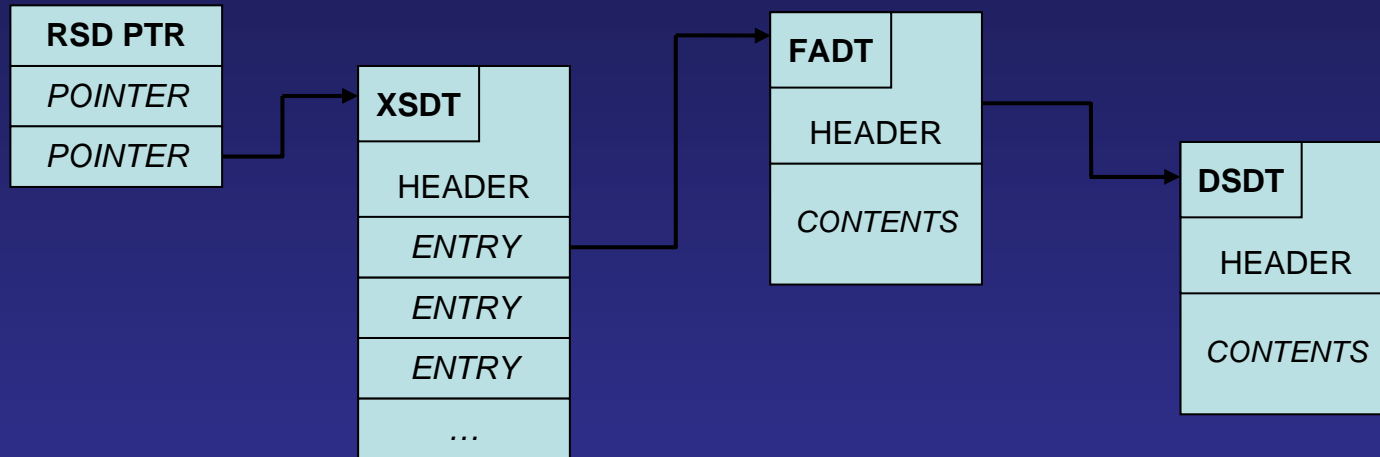
# Extra Material

# Typical ACPI Implementation





# Key Tables



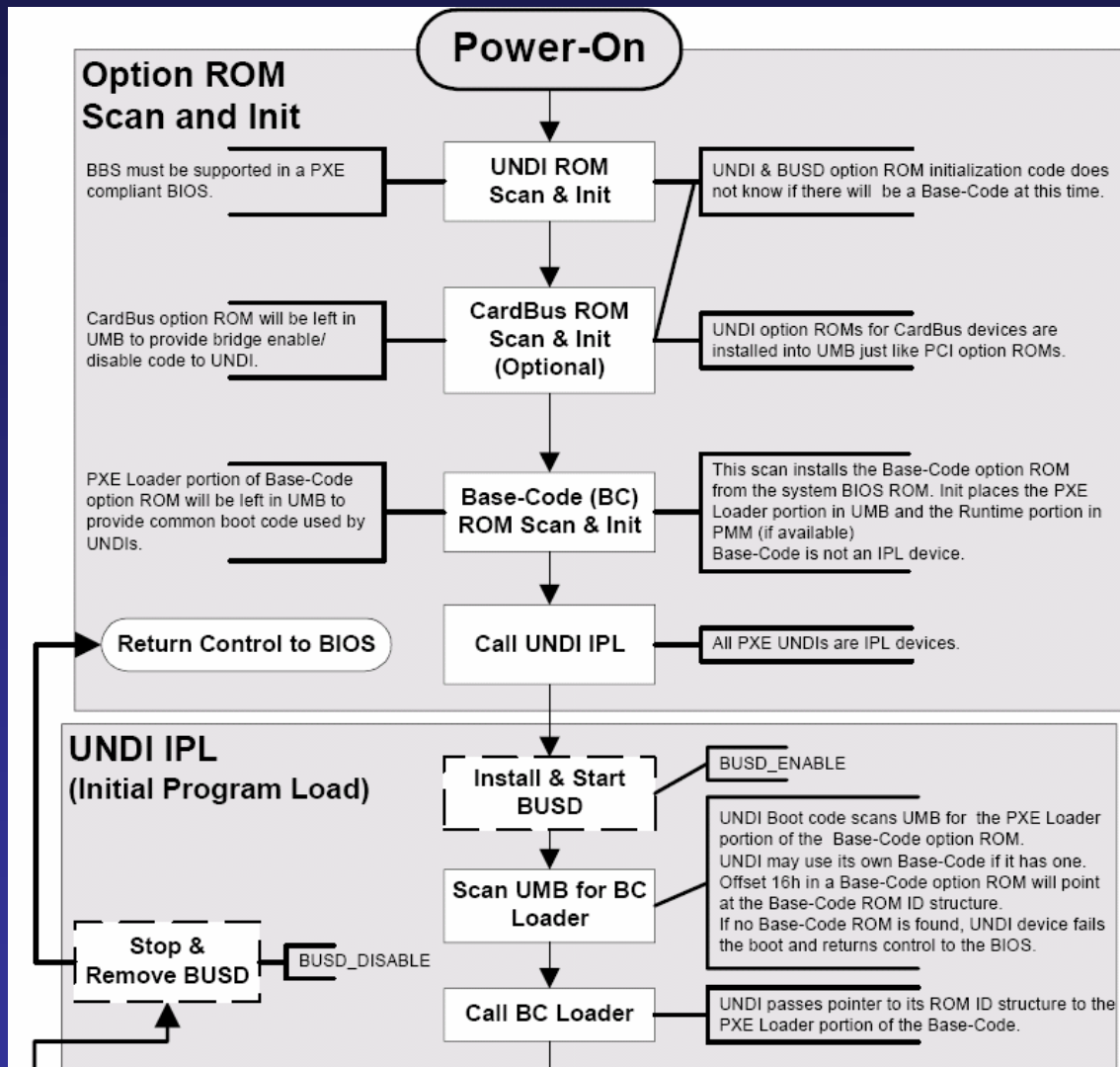
# PCI Expansion ROMs (2)

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

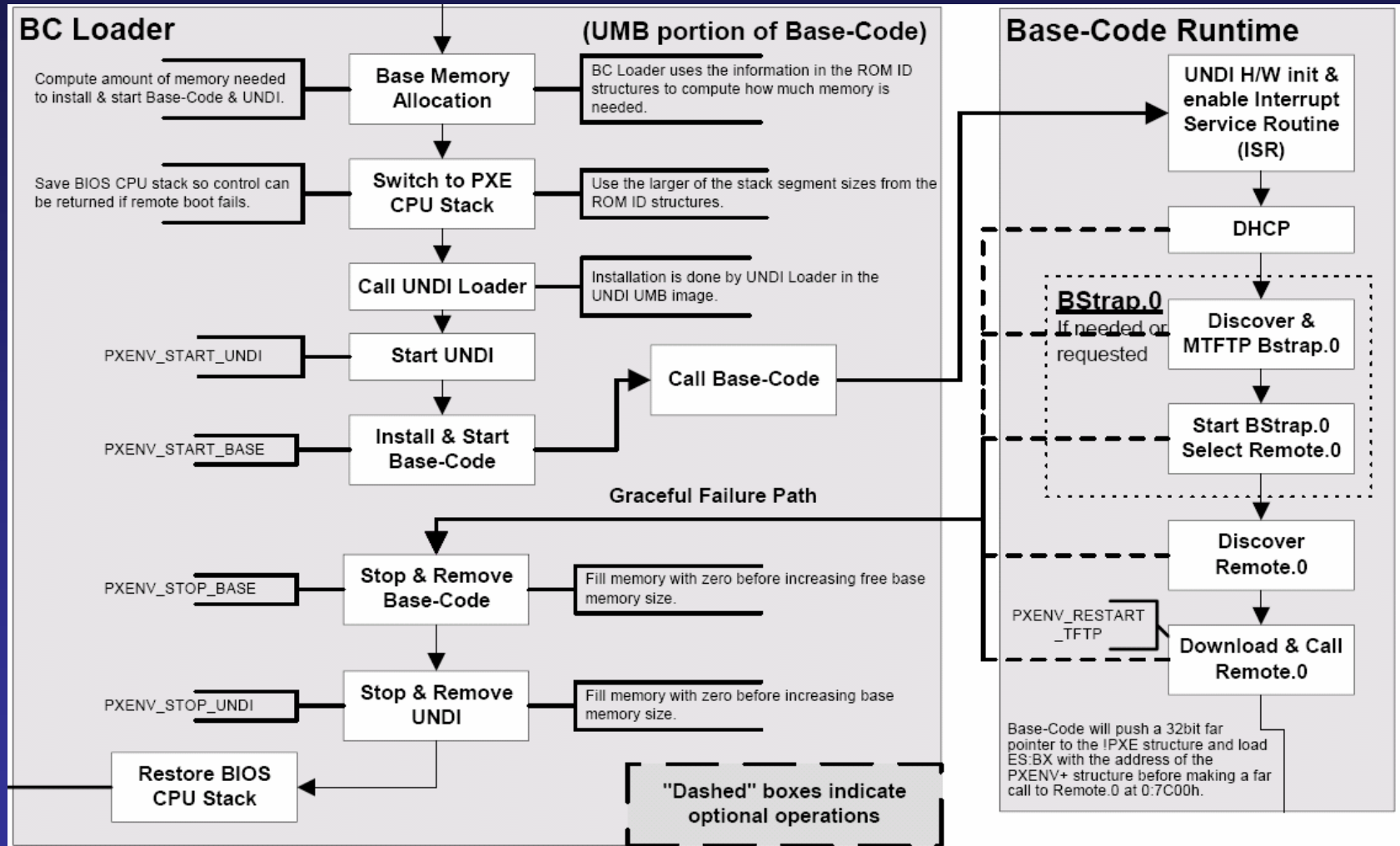
# Breakpointing interrupts in Bochs

- Bochs is an Open Source x86 emulator
- It contains an integrated debugger, or can be used with GDB
- We can create and debug our own expansion ROMs
- And breakpoint execution of real mode interrupts
- Demonstration: Windows and int 10h

# PXE ROMs (1)



# PXE ROMs (2)



# Malicious ROM Detection (3)

- Write 1's to Bits 21-32 to retrieve Expansion ROM Base Address length
- Allocate memory
- Set Expansion ROM Enable bit
- Set Memory Space bit in Status Register

# PCI Configuration Space

31		16		15		0		
Device ID				Vendor ID				00h
Status				Command				04h
Class Code					Revision ID			08h
BIST	Header Type	Latency Timer	Cache Line Size					0Ch
Base Address Registers								10h
								14h
								18h
								1Ch
								20h
								24h
Cardbus CIS Pointer								28h
Subsystem ID				Subsystem Vendor ID				2Ch
Expansion ROM Base Address								30h
Reserved								34h
Reserved								38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line					3Ch

# Misconception #1: Development Difficulty

- Remove hardcoded addresses:
- Provide update mechanism:  
New objective: a low level, pre-boot update method...



# Misconception #2: Firmware Differences

- Remove hardcoded addresses:
- Provide update mechanism:  
New objective: a low level, pre-boot update method...

# Misconception #3: Deployment Difficulty

- Remove hardcoded addresses:
- Provide update mechanism:  
New objective: a low level, pre-boot update method...