Virtualisation security and the Intel privilege model

Tavis Ormandy, Julien Tinnes
Google Inc.

Agenda

- Explain **core** x86/x86_64 virtualisation techniques
  - Focus on CPU virtualisation
  - Not on virtualisation of other devices
- Show various examples of exploitable bugs
  - In CPU virtualisation only

Virtualisation security (VS)

- Ideally
  - different guests behave like different physical machines
- It's an obvious requirement for many use cases
  - Workload consolidation
  - Workload isolation
  - Testing

VS: isolation

- Isolation is a fundamental requirement
  - Popek and Goldberg in 1974
- Secure isolation of virtual machines
  - Guest → Host
  - Guest → Guest
- Host → Guest is ok (and usually trivial)

VS: equivalence

- From Popek and Goldberg:
  - *A program running under the VMM should exhibit a behavior essentially identical to that demonstrated when running on an equivalent machine directly*
- Virtualisation needs to be *correct*
  - Should not introduce new flaws to guest OS

VS: detection

- Detecting a virtual machine
  - Possible, usually quite easy
  - Could be made impossible in theory?
    - No-one will probably ever have an incentive to do this
  - More and more useless as virtualisation becomes mainstream
- Not mentioned anymore in this talk

VS: aspects

1. Core system emulation (CPU, memory etc...)
2. Devices emulation
3. Guest to Host communications

(2) and (3) previously discussed

We focus on a part of (1) today

VS: previous work

- Mostly things running in Ring3 on the host
- Bugs in Guest/Host communication
  - VMware shared folders (IDefense, CORE)
- Bugs in devices emulation
  - VMware NAT Networking (Tim Shelton)
  - 'Bitblt'-style bugs in video emulation (Tavis, Rafal, Kostya)

Bitblt-style bugs*

- BLT = block transfer on PDP-10
- BITBLT refers to any algorithm to copy rectangles of bits on a bit-mapped device
- Bugs in display devices emulating code
  - Complex devices to emulate
  - Bitblt non trivial

Bitblt-style bugs

- QEMU VGA device (Tavis Ormandy)
  - QEMU device emulation used in Xen/Virtualbox as well
- Xen's para-virtual framebuffer (Rafal Wojtczuk)
  - Check ITL's paper
- VMware Cloudburst-related bugs (Kostya Kortchinsky)
  - See cool hacking in 3D

Bitblt-bugs exploitable from guest Ring3 ?

- Most device emulators should only be exploitable from guest kernel
- In some cases, enough control from Ring3
  - In some display devices, lots of pass-through from Ring3 to the device emulator
  - And from device emulator to the host's driver
- Remember remote Nvidia vuln ? (CVE-2006-5379, Rapid7)
  - From a web page visited in Guest to Host Ring0 ?

VS: Core system/CPU emulation*

- Two aspects
  1. Privilege elevation in Guest
  2. Guest to Host escapes?
- Not many public bugs
  - Quite well tested area (every OS is a potential fuzzer)
  - Complex area, hard to debug

X86 virtualisation

- A challenge
  - Not virtualisable in Popek and Goldberg's sense
  - Described by John Scott Robin and Cynthia E. Irvine in 1999
- Often poorly understood
  - Essentialy driven by VMWare (closed-source) for many years
  - Requires good low-level understanding

X86 virtualisation today

- Available today (for both x86 and x86_64)
  - Full virtulisation (Bochs, QEMU)
  - Paravirtualisation (Xen, VMware)
  - VMware-style (VMware, VirtualPC, VirtualBox...)
  - Hardware virtualisation (Xen, VMWare, VirtualPC, VirtualBox, KVM...)

Full virtualisation

- Started with Bochs (simulation)
- QEMU uses dynamic translation to make it fast
- Principle
  - Emulate devices in userland
  - Emulate the CPU by translating native instructions to instructions for the host CPU
- Is not Popek and Goldberg virtualisation

Trap&Emulate virtualisation strategy

- VMM runs with full privileges (Ring 0)
- Run the guest kernel at lower privilege
  - Privileged instructions trap
  - VMM catches the trap and emulates the instruction
- Run userland code ”as is”
- Somehow find a way to isolate kernel code from userland code
  - ideally other privilege level if available

X86 challenges

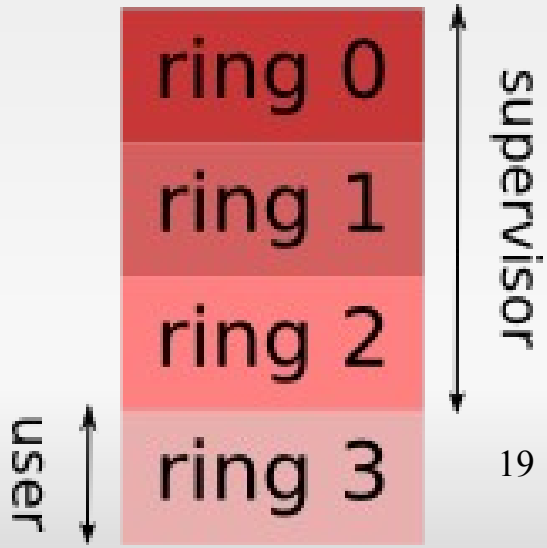
- 17 instructions don't meet Popek and Goldberg criteria
  - Detailed by K. Lawton and S. Robin
- Non faulting access to privileged state
  - SGDT, SLDT, SIDT, SMSW, PUSHF...
- Behave defferently at lower privileges w/o trap
  - POPF, LAR...

Binary translation

- Introduced into VMware in 1999
  - Now used by Virtual PC, VirtualBox...
- Deprivilege the kernel to execute in ring 1
- Dynamically modify kernel code to overcome limitations
  - BT translating all kernel code is slower
  - But offers lots of opportunity to optimize things (prevent expensive traps)
  - They managed to make this fast

VMware-style: CPU (1)

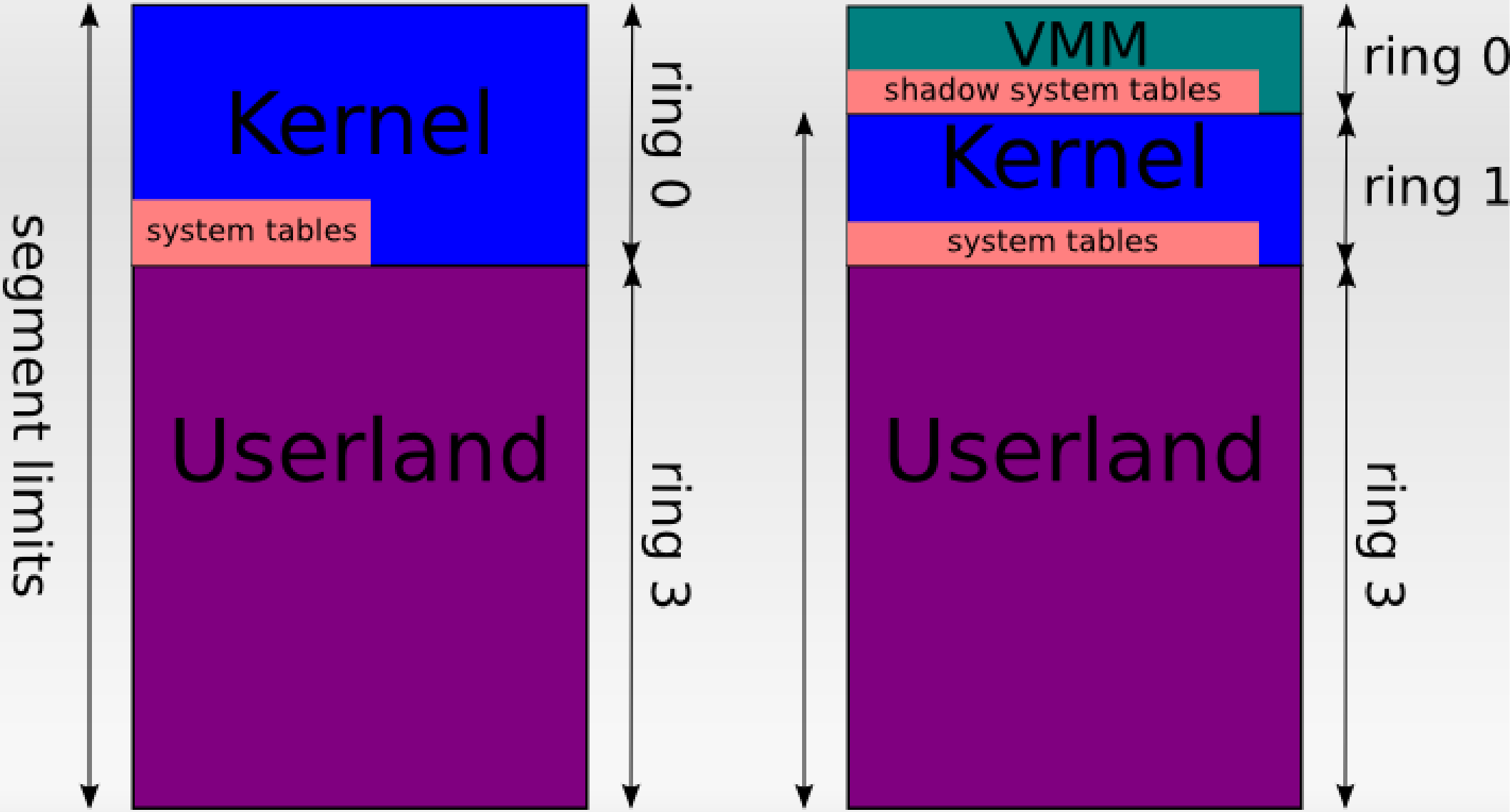
- VMM runs in Ring0
- Device emulation runs in Ring3 on host
- Guest kernels run in Ring1
  - protected from guest Ring3 by pagination as usual
- Pagination can not isolate the Ring 0 VMM from ring1 (ring{0,1,2} = supervisor)
  - Use segmentation instead



VMware-style: CPU (2)

- Ring3 mostly runs ”as is”
- You can access privileged states from Ring3
  - SGDT, SIDT, (SLDT) reveals locations of the real tables (inside the VMM, on top of address space)
    - Real tables = shadow tables
    - This is the true explaination for redpill

VMware-style virtualisation



VMware-style security and BT

- BT takes care of GS: overrides. GS segment can access VMM!
- If BT confusion, instant ring0 (guest → host escape!)
  - VMware's BT seems of good quality
  - What about others?
- BT is fragile and can be broken by CPU errata
  - Not very well studied
  - Stay tuned

VMware-style security: more

- Shadow paging complexities
- Handling all subtleties like a real CPU is complex
  - What we called ”correctness” before
  - Mostly leads to guest privilege escalation
  - We will show examples of those
  - Can be seen as ”Virtual CPU errata”
    - And could be worked-around by OS in theory

Augmenting VMware-style's attack surface

- Full emulation mode not very often used
  - But we can reach it from guest by using ED segments
- Many other things are not done by regular kernels
  - attacks should be conducted from Ring1
  - Or with IOPL != 0
  - Yet it's much easier to focus on Ring3

Ex1: VirtualPC instruction decoding*

- Tavis Ormandy, Julien Tinnes (CVE-2009-1542)
- Some privileged instructions could be executed from Ring3
- wbinvd, clts execute in cpl > 0
- rdpmc ignores cr4.PCE
- Explanation:
  - They do fault, but VirtualPC catches the exception
  - Wrongly checks the privilege and emulate the instruction

Ex1: VirtualPC clts decoding exploitation (1)

- clts clears cr0.TS
- The TS flag is set on task-switches
- The TS flag is tested on every executed FPU instruction by the processor. If set, raise #NM
- Most OS don't use hardware task switching
- They handle task switches in software
  - And set cr0.TS manually, but only if needed (if previous process used FPU and the flag got cleared)

Ex1: VirtualPC clts decoding exploitation (2)

- If you unexpectedly clear TS by using this bug, it will be forever unset
  - No FPU instruction will ever trap
  - The operating system will never know that any FPU instruction occured
  - All processes will share the same FPU state
- Did we say FPU ?
  - We mean FPU/MMX/SSE/SSE2/SSE3/SSSE3/SSE4

Ex2: VMware, Trap Flag Set by IRET Not Cleared for CCh Instruction*

”The Trap Flag persists across the mode switch when a single-byte "INT3" instruction (CCh only, not CDh/03h) executes, if the Trap Flag was set by a kernel-mode IRET”

- Derek Soeder (CVE-2008-4915)
- User code can cause an exception at the very first instruction of the INT3 handler
- Some kernels may rely on this to never happen for security
  - Windows 64 expects a particular prologue executed if an exception occurs from Ring0
  - Otherwise, kernel's GS register stays user-controlled

Ex3: VMware Mishandled Exception on Page Faults*

- Tavis Ormandy, Julien Tinnes (CVE-2009-2267)
- VMware advisory published last week
- Bug in the core CPU emulation

Ex3: Page Fault Exceptions

- A page fault exception occurs when code...
  - Attempts to access a non-present page
  - Has insufficent privilege to access a present page
  - Other paging related errors
- The handler is passed a set of flags describing the error

  - `I/D` – Instruction / Data Fetch
  - `U/S` – User / Supervisor Mode
  - `W/R` – Read / Write access
  - `P` – Present / Not present

Ex3: Supervisor Mode

- If the processor is privileged when the exception occurs, the supervisor bit is set
- Operating system kernels can use this to detect when special conditions occurs
- This could mean a kernel bug is encountered.
  - Oops, BugCheck, Panic, etc.
  - Or an unusual low-level event
- Can also happen in specific situations (copy-from-user etc...)
- If the processor can be tricked into setting the flag incorrectly, ring3 code can confuse privileged code.

Ex3: VMware Invalid #PF Code

- We found a way to cause VMware to set the supervisor bit for usermode page faults.
- Far calls in Virtual-8086 mode were emulated incorrectly.
  - When the cs:ip pair are pushed onto the stack, they are done so with supervisor access.
  - We were able to exploit this to gain ring0 in VMware guests.
- Linux checks for a magic CS value to check for PNPBIOS support, we were able to use this feature to redirect kernel execution to NULL.
  - But, because we're in Virtual-8086 mode we must be permitted any value cs.

Ex3: VMware Exploit

- We mmap() shellcode at NULL, then enter vm86 mode
  - We found a separate vulnerability to bypass mmap_min_addr (CVE-2009-1895)
- When we far call with a non-present page at ss:sp, a #PF is delivered.
- Because we can spoof arbitrary cs, we set a value that the kernel recognises as a PNPBIOS fault.
  - The kernel tries to call the fault handler.
  - But because this is not a real fault, the handler will be NULL
- r00t :-)
- Demo!

Paravirtualisation: Xen*

- Instead of doing BT on the guest kernel, require the guest kernel to be modified
- Uses Ring deprivileging as well (and VMM on top of address space)
- The kernel performs hypercalls to the hypervisor

Paravirtualisation vs. BT

- VMware-style: the guest kernel (in Ring1) is under tight BT control
  - Uses **both** BT and segmentation to protect VMM
  - BT can try and prevent Ring1 code from performing attacks
  - But arbitrary Ring1 code = instant ring0 (access to VMM)
- Paravirtualisation
  - Arbitrary Ring1 doesn't imply arbitrary Ring0
  - But Ring1 is contained by segmentation only

64 bits virtualisation (1)

- AMD dropped segmentation
  - Many cool security features impossible on x86_64
    - PaX' UDEREF (and others) kernel protections
    - NativeClient would be very different in 64 bits
- We will miss it forever
- Hardware virtualisation supposed to make up for this
  - More on this later

64 bits virtualisation (2)

- How would you prevent Ring1 from accessing VMM ?
- VMware
  - Tighter, (way) more complex BT ?
  - Never implemented as far as we know
- Xen
  - Put guest kernel in Ring3. (Ring compression)
  - Full address space switching to protect guest kernel
  - Big performance hit
    - TLB cache filter buggy and deprecated

64 bits: AMD brings back segmentation

```
64-bit Segment Limit Check Mechanism:
- Assume segment-addressed access of form SEG:ADDR
- if ( 64bit_mode && EFER[13] && (CPL > 0) &&
        (SEG==DS || SEG==ES || SEG==FS || SEG==SS) )
     { limit = (SEG.G ? (SEG.limit << 12)+0xFFF :
0xFFF)));
          if (ADDR > ((0xFFFF << 32) + limit))
             generate_std_segment_limit_GP_fault();
      }
```

- Very secretive (still not in official doc)
- Bare minimum for VMware
  - No CS check (code offset controled by BT anyway)
  - No GS limit check
    - VMware rewrites GS-overrides and uses them to access the VMM.

64 bits virtualisation impact

- Long mode supports 64 bits and compatibility (32 bits) submode
- A 64 bits operating system typically supports both
- 64 bits adds complexity
  - Example: far call to 32 bits code segment in a 64 bits process on a 64 bits kernel on a 32 bits host
  - No, there is no typo
- Address space switching on Xen non trivial
  - Any optimization on this might introduce exploitable bugs

Ex4: VMware, Interrupt Can Occur at Non-Canonical RIP After Indirect Jump

- Derek Soeder (CVE-2008-4279)
- In 64 bits, there are canonical and non canonical addresses
  - 48-bits addresses (sign extended to 64 bits)
- jmp [mem] to a non canonical location will #GP at jmp instruction
- In VMware, only the next one would #GP
- Exploitation
  - Windows 64 expects a particular prologue executed if an exception occurs from Ring0 and a particular epilogue has not executed yet
  - Using this, you can make the #GP handler #GP on iretq
  - The kernel will use the restored user-controlled GS

Hardware virtualisation*

- Fast and secure virtualisation on IA32 is challenging
- Without segmentation, x86_64 would be harder and slower
  - (AMD brought segmentation back on AMD64 for VMware only later)
- Hardware virtualisation allows the architecture to meet Popek and Goldberg's criterion
- Two incompatible designs, AMD SVM and Intel VT-x
  - Greatly lowers the bar to write an hypervisor

Hardware virtualisation (VT-x)

- Two new forms of CPU operation: VMX root and VMX non root
- VMM: root operation – Guest: non root operation
- Transitions: VM entry / VM exit
- Managed by a VMCS structure
  - VMCS also manages behavior in VMX non-root operation

Hardware virtualisation (VT-x)

- Popek and Goldberg compliant
- No Address space compression required
  - VMM can live in its own address space
- No ring compression
- No more non faulting access to privileged state
- No longer instructions that perform a different action in lower privileges w/ no trap

Hardware virtualisation (VT-x)

- Popek and Goldberg compliant
- No Address space compression required
  - VMM can live in its own address space
- No ring compression
- No more non faulting access to privileged state
- No longer instructions that perform a different action in lower privileges w/ no trap

Ex5: Virtual PC Vmexit Event Confusion

- Tavis Ormandy, Julien Tinnes (CVE-2009-3827)
- When a vmexit occurs, an exit reason is recorded in an MSR, which the monitor can then inspect
- Two interesting reasons are MOV_DR and MOV_CR
  - MOV_CR indicates the guest accessed a control register
  - MOV_DR indicates the guest accessed a debug register
- When the host decodes the reason for the exit, it can decide what to do, and then continue the guest.

Ex5: Virtual PC Vmexit Event Confusion

- The MOV_DR and MOV_CR events are *very* similar.
- It's tempting to handle them using the same monitor code, but there is an important difference
  - MOV_CR will check the guest cpl before vmexit
  - MOV_DR *will not check cpl.*
- VirtualPC made this error in hardware virtualisation mode.
  - We can set the debug registers from ring3!
- This can easily be used for DoS (just make the guest kernel double fault), but there may be more attacks (DR7?)

Ex6: KVM Vmexit Event Confusion

- The same bug was found in KVM (CVE-2009-3722)
- Found (independantly) by Avi Kivity in september 2009

```
static int handle_dr(struct kvm_vcpu *vcpu, struct kvm_run *kvm_run)
         unsigned long val;
         int dr, reg;
+        if (!kvm_require_cpl(vcpu, 0))
+                return 1;
         dr = vmcs_readl(GUEST_DR7);
         if (dr & DR7_GD) {
```

- The Intel documention could be clearer on this point

Conclusion*

- The biggest attack surface to virtualisation is from the guest kernel
  - Device emulators and other Guest <-> Host communication
  - Bypassing binary translation in VMware
- Guest privilege escalation as a first stage
- Intel CPU virtualisation is complex
  - Inaccuracies can quickly lead to guest local privilege escalation (like CPU errata)
  - Including very small details

Thank you!

- Questions?

Appendix

Hardware virtualisation already existed on IA32

- VM86 to allow 8086 emulation
- Good introduction to hardware virtualisation
- Most of the code executes 'as is'
- The processor leaves VM86 though interrupts and exception
  - HW interrupt
  - IOPL-Sensitive instructions (CLI, STI, PUSHF, POPF, INT n, IRET)
  - Gives a chance to the VM86 monitor to emulate them

VMware-style: shadow paging

- The VMM needs to virtualize memory access
- The guest maintains primary structures
- The VMM maintains shadow structures
  - as seen by the processor
- There is not a 1:1 mapping between them
  - The shadow structure can be viewed as a cache of primary structures
- The logic leaves room for optimization
  - And can be complex

More hardware virtualisation

- VT-d (IOMMU)
- Hardware virtualisation is actually slower than VMware-style virtualisation in many use cases
  - VMM intervention on guest context switches
  - VM exits are very expensive
  - This should be solved by nested paging (Intel EPT)

Ex: VMCI priv escalation

- VMSA-2009-0005

Ex: KVM hypercalls

- CVE-2009-3290

Ex: Windows VDM Zero Page Race Condition Local Privilege Escalation Vulnerability

- Derek Soeder (CVE-2007-1206)

Ex: Xen debug register handling

- Jan Beulich (CVE-2007-5906)