

matasano



symantec™



root labs

Don't Tell Joanna

The Virtualized Rootkit Is Dead

Nate Lawson @ Root Labs
Peter Ferrie @ Symantec
Thomas Ptacek @ Matasano



matasano



symantec™



root labs

Introduction

What We're Going To Talk About

- ★ HVM Malware Recap
- ★ Nothing Is 100% Undetectable
- ★ Samsara: A Framework For HVM Malware Detection
- ★ Conclusion: A Cat And Mouse Game



matasano



symantec™

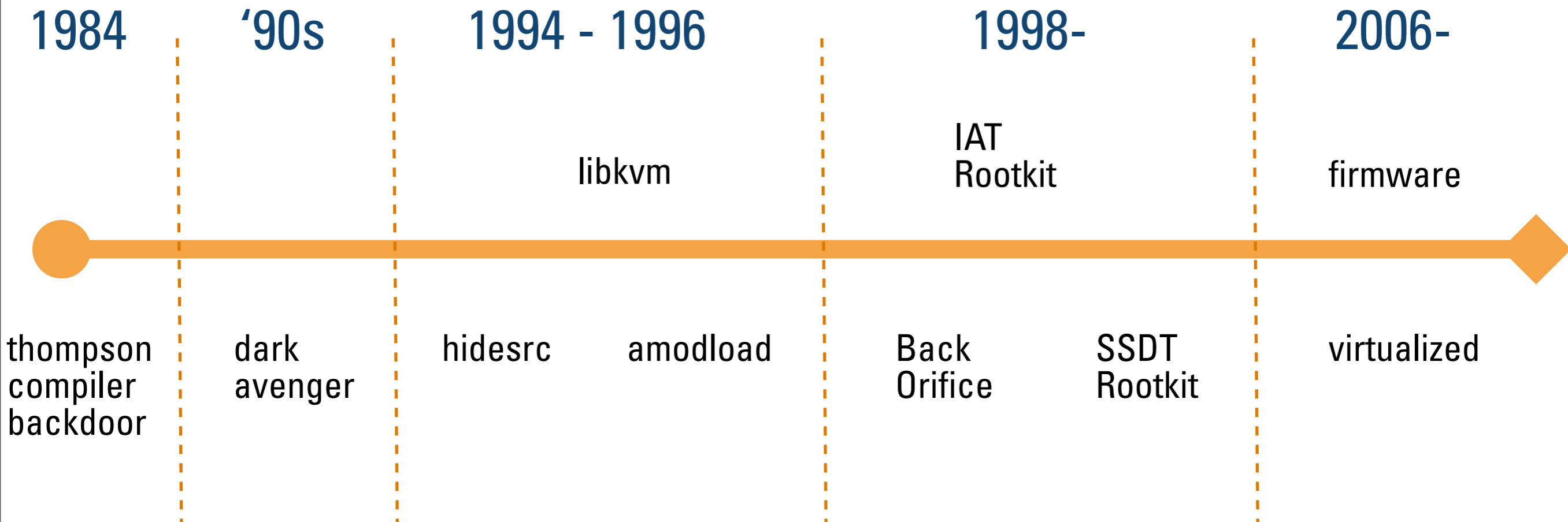


root labs

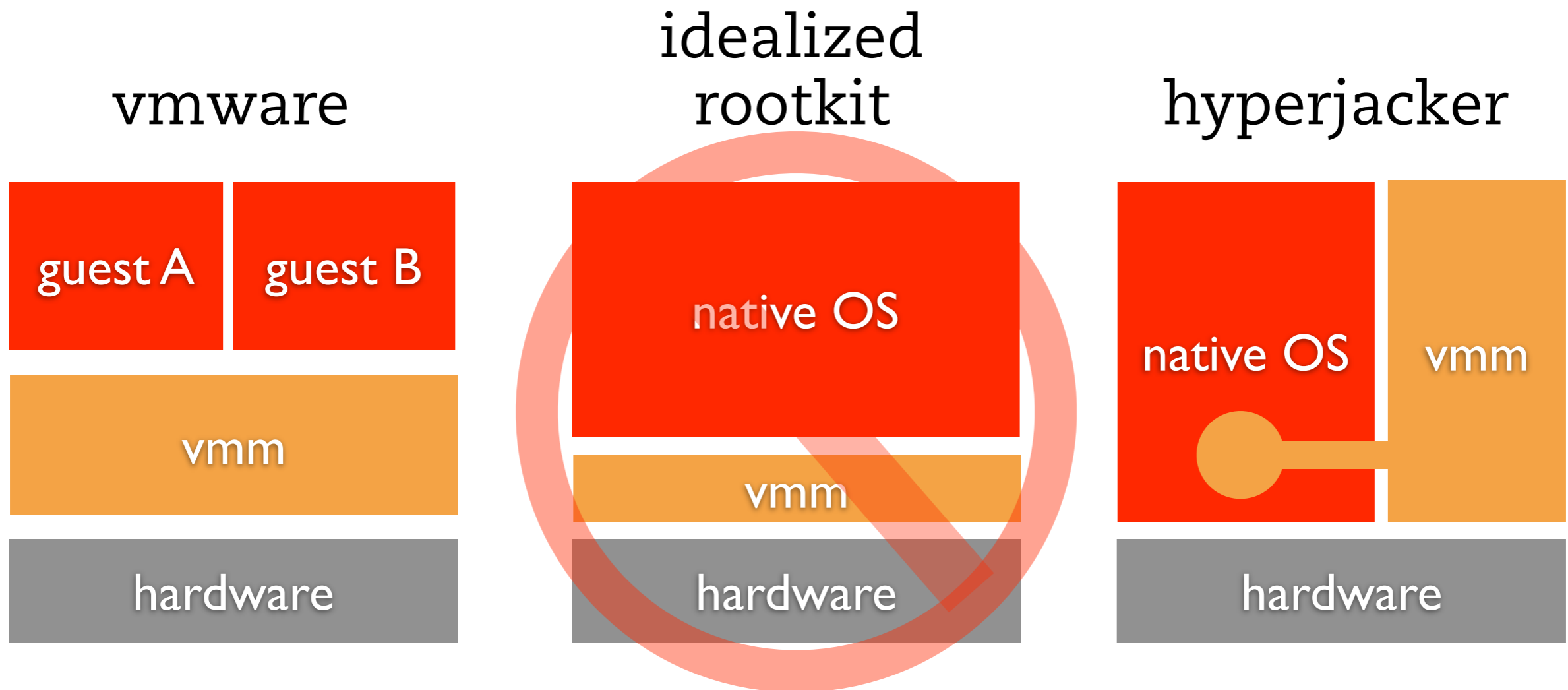
HVM Malware

A Remedial Course

We've Seen This Movie Before



Hyperjacking vs. Virtualization



small footprint = direct guest hardware io

Hyperjacker Analogy: WebScarab

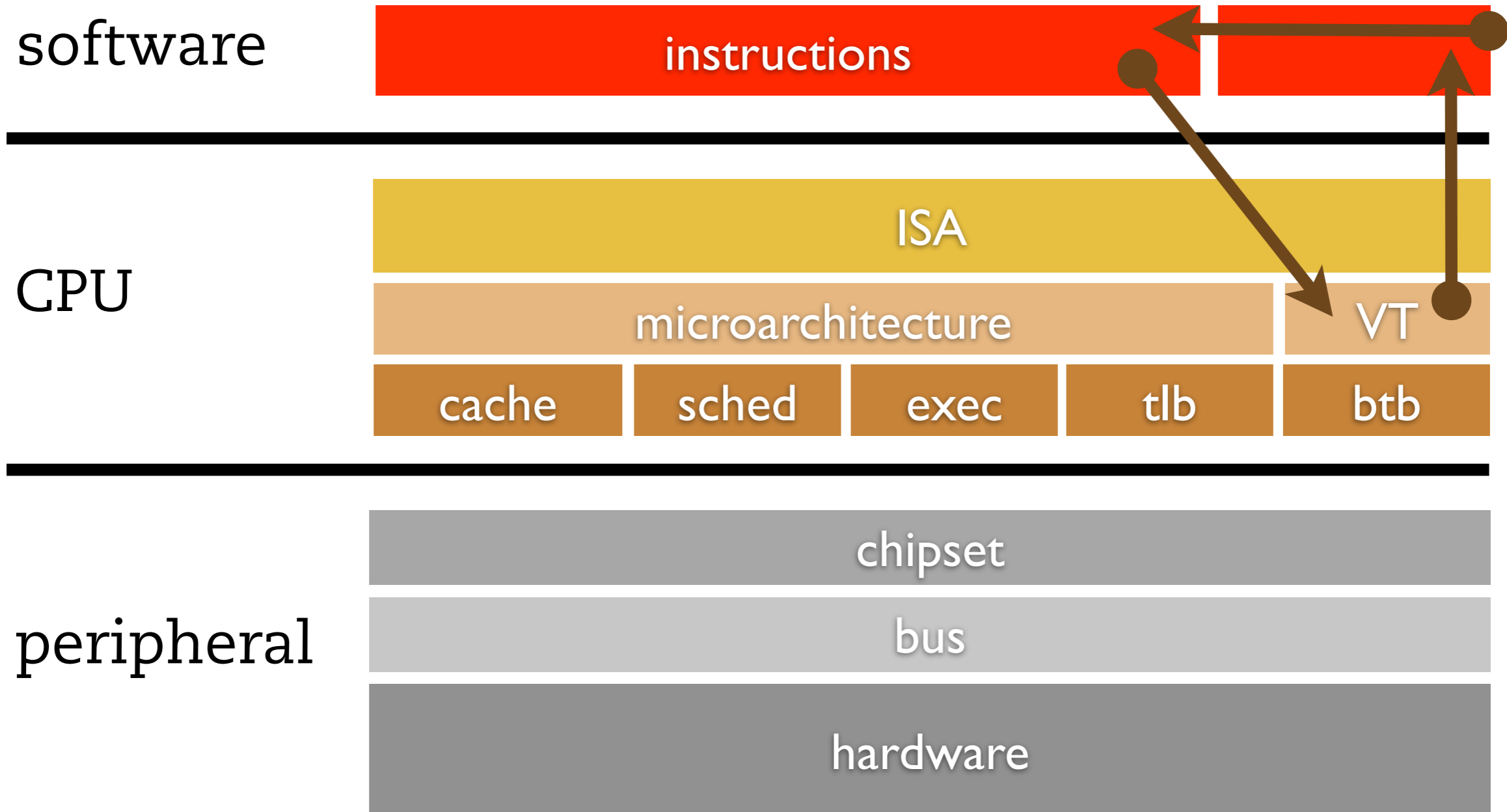


Almost all hardware functionality is left untouched. VMM picks and chooses (via trap handler) what to manipulate.



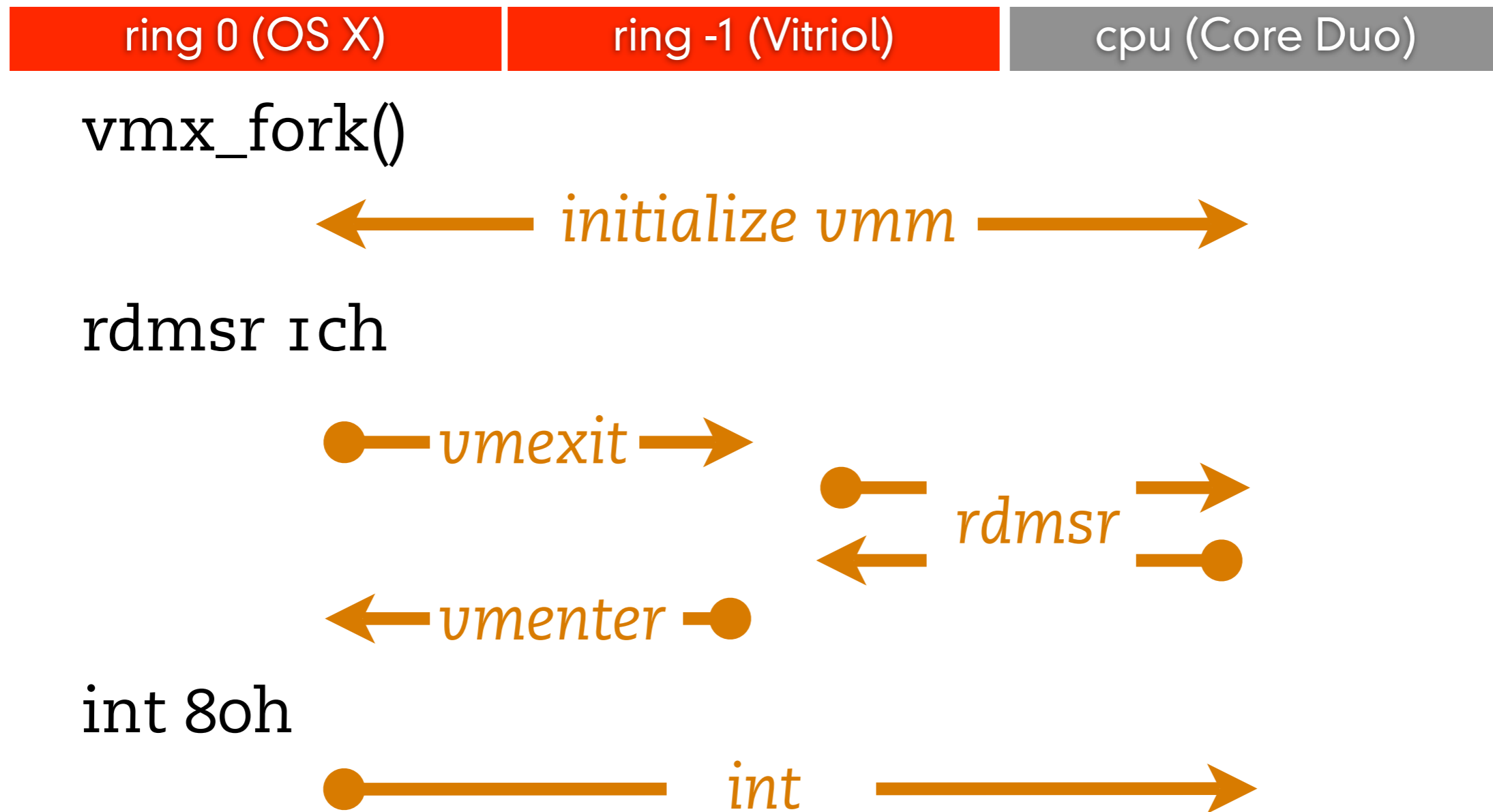
Vitriol 1 is less than 2000 lines of C code.

X86 System Hierarchy



Vitriol: ddz's HVM Hyperjacker

★ Dino Dai Zovi's 2006 Matasano Black Hat Talk



Blue Pill: Joanna's HVM Rootkit

★ Joanna Rutkowska's 2006 COSEINC Black Hat Talk

★ Just Like Vitriol, *but*:

- *uses AMD SVM, not Intel VT-x*
- *Vista, not OS X*
- *loads self via Vista (beta) swap bug*
- *implements network IO with debug registers*
- *loads LWIP stack into the kernel*
- *apparently implements nested virtual machines*

★ Claim: *100% Undetectable Malware*



matasano



symantec™



root labs

Nothing
is 100% Undetectable

Detecting VMWare Is Easy

★ Unrealistic outdated device hardware

- *ISA ethernet controller?*
- *440BX chipset?*

★ Holes in virtualization

- *SIDT*
- *Microsoft-manufactured motherboard*
- *Registry keys*
- *“VMware” in video and SMBIOS strings*

★ Guest-to-host communication channels

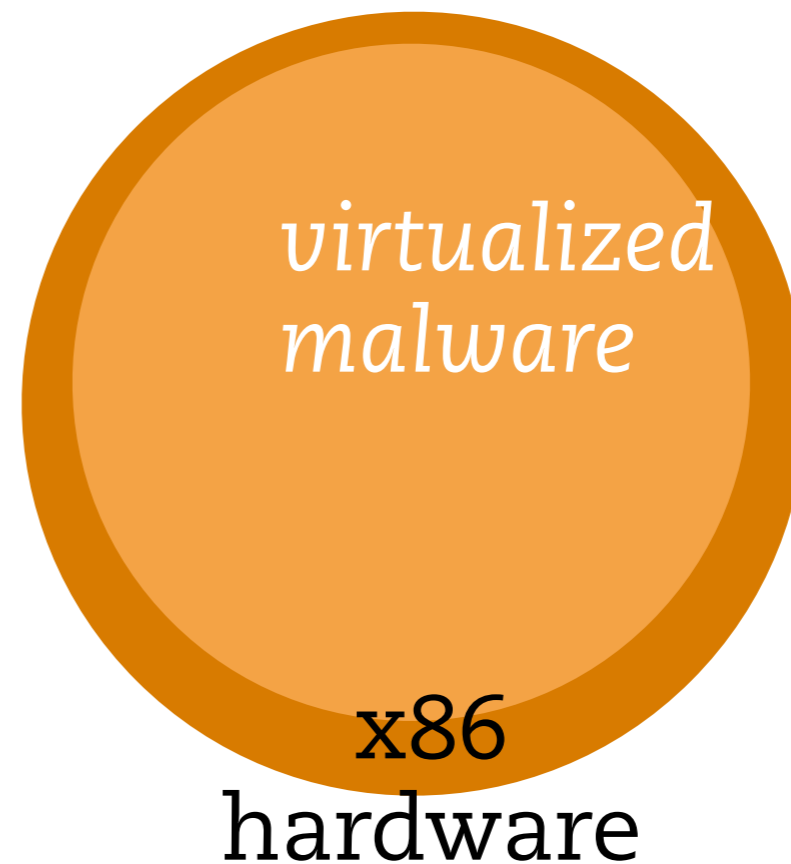
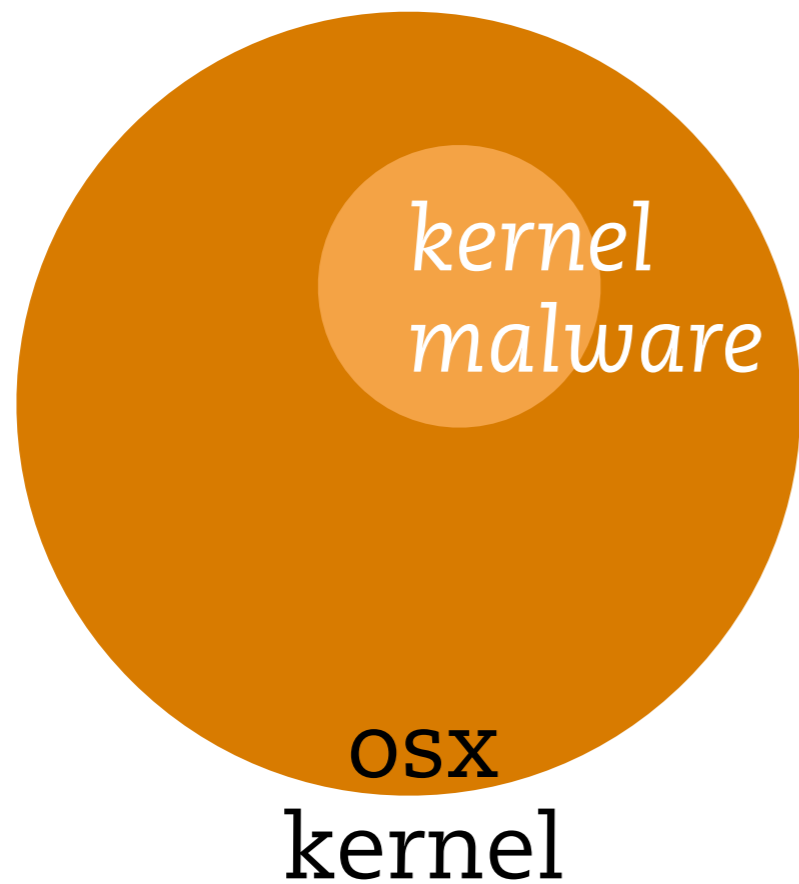
- *VMware: inb/outb to magic port*
- *VirtualPC: illegal instructions*

★ Wide timing variances

- *Hard to trap timer reads (RDTSC), accuracy suffers*

Cross Section

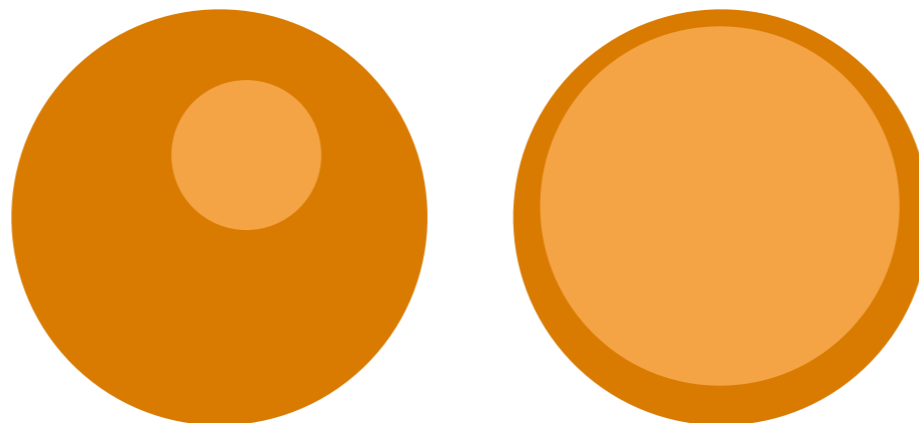
A measure of stealth, quantified by intrusiveness.



Cross Section for Virtualization

cross section • *noun*: amount of the original system that rootkit must emulate to remain hidden • *etymology*: radar, stealth planes

- ★ Varies by layer chosen for rootkit
- ★ Dictates complexity of rootkit
 - *fails to trap and emulate a feature: detectable*
 - *emulation too complex: big target, detectable*
- ★ **“Entire x86 hardware platform” is a huge cross-section**



Fundamental Problem

This instruction:

cpuid

should take 200 cycles, not 5000, is unprivileged, and should have no impact on cache, BTB, or TLB.

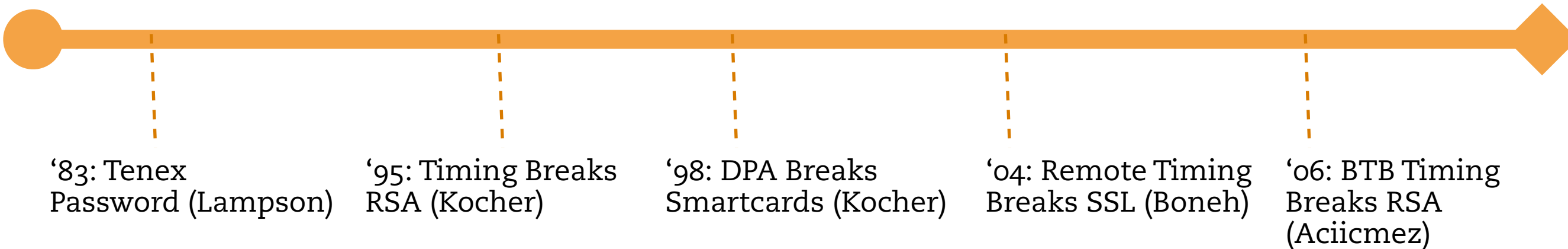
Three Detection Strategies

- ★ Strategy 1: Side-Channel Attacks
VM overhead creates detectable “trails” through microarchitecture that are prohibitively hard to conceal.
- ★ Strategy 2: Vantage-Point Attacks
VM cross-section forces it to recognize and emulate many obscure hardware features.
- ★ Strategy 3: Vulnerability Analysis
The more features a rootkit implements to hide itself, the more bugs it exposes.

What Is A Side Channel Attack?

- ★ Any resource consumed in a logic-dependent way leaks information. *For example:*

```
if (strcmp(guessPassword, realPassword))  
    return LOGIN_FAILED;
```



'83: Tenex
Password (Lampson)

'95: Timing Breaks
RSA (Kocher)

'98: DPA Breaks
Smartcards (Kocher)

'04: Remote Timing
Breaks SSL (Boneh)

'06: BTB Timing
Breaks RSA
(Aciicmez)

But Real Systems Are Too Noisy!

BZZZZZT!

- ★ *Astounding: “Opportunities and Limits of Remote Timing Attacks”; Crosby, Riedi, Wallach*
 - *WAN timing: 15-100 microseconds resolution (!)*
 - *LAN: 100 nanoseconds*
- ★ **If noisy, take more samples and average**
 - *Decouples noise from true resource consumption*
 - *Local access = higher resolution*
- ★ **We monopolize ring 0 for several microseconds**
 - *Less than an AV scanner*
- ★ **Data-dependent side-channels are *hard to eliminate*.**

Finding Side-Channel Attacks

★ How to look for it:

- *Enumerate all resources your opponent has to use*
 - say, “executing instructions”
- *Identify how to measure that resource from your vantage point*
 - Branch target buffer state can be read by timing branches in your own thread
- *Take as many measurements as possible*
- *Eliminate jitter with Stat 101*

★ Secret: identifying whether or not any code (i.e., hypervisor) executed *much* easier than extracting a key from that code

Vantage Point Attacks

- ★ Dilemma: either let me talk directly to the hardware, *which will betray you*, or emulate the hardware, *with perfect fidelity*.
- ★ HPET: alternative high-precision timers (supercedes but does not eliminate the RTC).
- ★ Performance Event Counters: instructions retired (in/out of VMM), cache misses, branch mispredictions, model-specific events.
- ★ GART: scatter-gather memory map for graphics devices

Finding Vantage Point Attacks

- ★ Embedded timers (exposed directly, via CSR, or indirectly via behavior). *Force Blue Pill to emulate every mainstream peripheral.*
- ★ Model-specific MSRs and CSRs (particularly scattered in sensitive functionality). *Not all MSRs and CSRs are documented.*
- ★ “Bounceable” Memory Access through devices (DMA rings, etc). *Force Blue Pill to emulate every mainstream peripheral.*


Finding Hypervisor Bugs

★ Sources of Bugs

- *nested/VTX*
- *errata*
- *vtx loading errata*

★ Where To Look

- *Get all datasheets and errata*
- *Descriptions give microarchitectural behavior (i.e., priority of faults in various error cases)*
- *Focus on items that require a full simulator to emulate correctly*
- *Or, behavior that can't be trapped/emulated*
- *Special credit: "won't fix" errata*



talk about
vmware
interrupt bug



matasano



symantec™



root labs

Samsara

*a framework for detecting
virtualized malware*

Implementation overview

- ★ three tests implemented

- *Instruction and data TLB*
- *HPET*
- *VT Errata*

- ★ test framework

- ★ rantipole: HVM rootkit simulator

- ★ futures

What's a TLB?

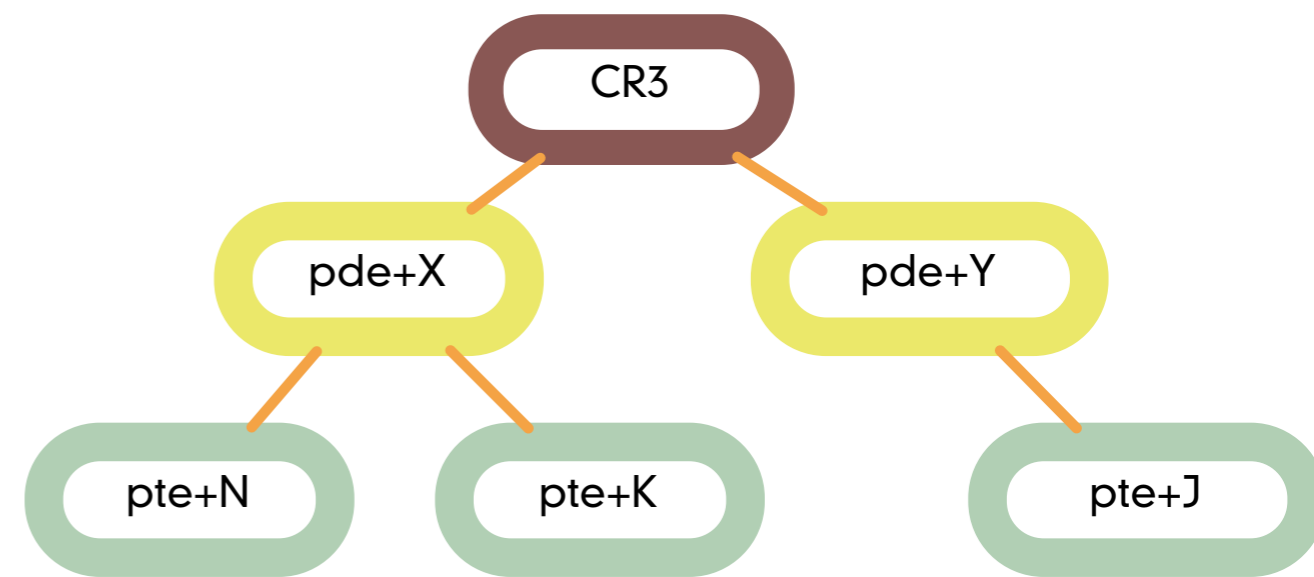
Translation Lookaside Buffer

–Remembers the translated addresses of memory you touched

- Like a cache for page tables (Virtual memory 101)
- Not directly visible from software but you can affect it
 - Flush it completely (*MOV CR3/CR4*)
 - Flush an individual page (*INVLPG*)
 - Indirectly by R/W/X from a page

bfff430h

1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0



When a VMEXIT occurs, hypervisor execution and memory access leave trails through the TLBs

General Approach To Snooping

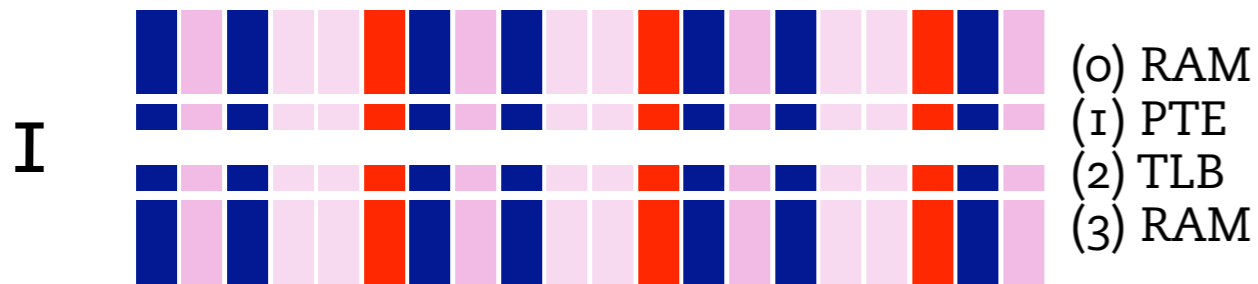
This sequence:

desync tlb
color memory
cpuid
read memory

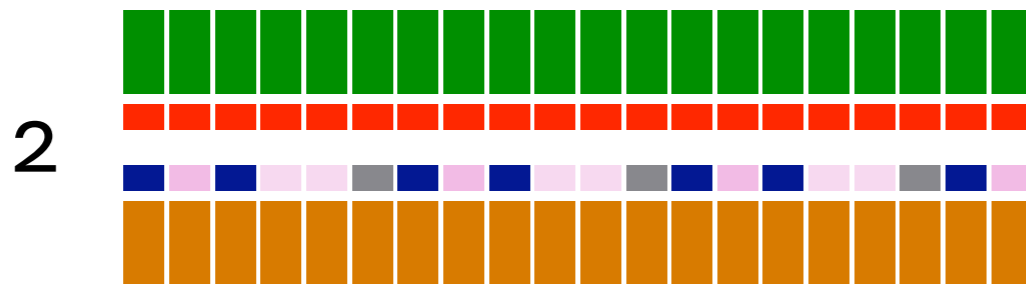
should be identical to
this sequence:

desync tlb
color memory
nop
read memory

TLB Snooping



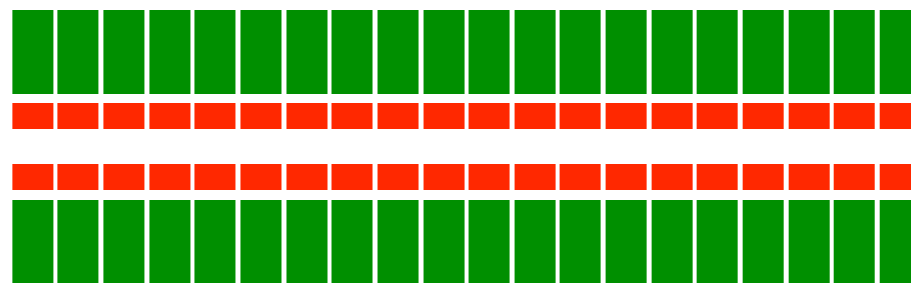
pte, tlb synced,
“random”



pte, tlb desynced,
RAM colored

post vmexit:
basic VT-x

3(a)



post vmexit:
AMD SVM

3(b)



Aside: ASID's Don't Help, Joanna

- ★ “But guest TLB usage can be partitioned with ASID”
 - *Application Space ID, tag that says which context owns a TLB entry*
- ★ Made for performance, not transparency!
- ★ Hypervisor still needs to read exit condition from VM
- ★ VMEXIT trap handler needs to run
 - *Hence, at least one data and instruction TLB entry will be lost*
- ★ Hypervisor has to now flush TLB manually for guest
 - *Observable timing difference = side channel*

Data TLB Test implementation

- ★ Allocate $N+1$ pages
- ★ Fill N pages with "A", one page with "B"
- ★ Touch the pages in order to flush TLB and fill with those mappings
- ★ Remap N pages to one page without invalidating the TLB
- ★ Walk backwards through all N pages, reading data
 - *Keep a counter*
 - *Once you see "B", you've flushed the TLB*
- ★ Repeat test but add probe to trap to hypervisor
- ★ If count varies with trapping instruction, you're virtualized

Instructions Have Their Own TLB

★ Instruction TLB test mostly same

- *Pattern A = mov eax, 1; ret*
- *Pattern B = mov eax, 2; ret*
- *“Reading” pages by call \$PAGE, check value in eax afterwards*

★ Same design pattern works for other tests

★ Branch target buffer (BTB):

- *Modern x86 caches branch targets. Saturate the BTB cache and evictions will make certain branches take observably longer.*

★ Cache invalidation (INVD):

- *INVD flushes the cache but discards queued writes. Saturate the cache, cause VMEXIT, and then INVD out the whole cache; evictions will make certain writes persistent.*

Vantage Point Example: HPET

★ Clock VM exits via the HPET instead of the TSC.

★ Kernel:

- *export sysctl "hpet_attach"*
- *map HPET IO range to requesting processes*

★ Userland:

- *(1) sample hpet*
- *(2) sample tsc*
- *(3) cpuid (or other vmexit)*
- *(4) elapsed tsc*
- *(5) elapsed hpet*
- *(6) reconcile*
- *(7) repeat*

Other Timing Sources

- ★ The PIT (16 bit counter/interrupt timer)
- ★ The RTC (battery-backed realtime counter [1 sec granular] and interrupt timer)
- ★ The LAPIC (32 bit, device-dependent frequency)
- ★ The ACPI timer (24 bit PITx3 timer)
- ★ VGA blanking
 - *Fun drinking game: “count the exposed embedded timers in peripherals”.*

Bugcheck Example: VMCS Fuzzer

- ★ VMCS stores host (hypervisor) state, guest (OS X) state, execution controls, and status.
- ★ VT-x dox specify 50+ checks on fields, *checked in well-defined model-specific order*, generating different exceptions (GPF, SIGILL, VMMLAUNCH error, etc).
- ★ Break each check, verify expected exception/signal/result.

Extending Samsara

- ★ Plug-in interface for adding new tests
 - *Kernel or userland*
 - *Tests use configurable probe function*
- ★ Userland support programs
 - *Query and generate traces*
 - *Analyze results of traces*
- ★ Profiles/"signatures" for well-known hardware
- ★ Runs on MacOS and FreeBSD
 - *Easy to port to new archs*

Rantipole: An HVM Testbed

★ Rantipole is a (crippled) HVM detection testbed

- *only works on OS X*
- *only works on Core 1 Duo*
- *only works in native 32 bit mode*
- *loudly advertises itself*
- *stripped of any SMP sync code (UP-only)*
- *no “backdoor” or “malicious” capabilities*
- *self-destructs in 10 minutes*

★ Malware authors: you are better off reading Xen

What Rantipole Does

- ★ (1) check cpuid, feature msr for VMX
- ★ (2) allocate vmx and vmcs from IOMallocContiguous
- ★ (3) initialize vmcs, call vmclear
- ★ (4) copy segments, stack, cr3 to vmcs host and guest
- ★ (5) set host/root/hypervisor eip to trap handler
- ★ (6) set exec controls to pick events we want
- ★ (7) vmptld to add vmcs
- ★ (8) (a) vmlaunch (b) spin

Futures

★ Implement more tests

- *VGA blanking interval timer*
- *BTB*
- *INVD*
- *Multi-core cooperating threads*

★ Improve simulator to validate tests

★ Port to new archs

★ None of this is really important anyway

tom: BAT
SIGNAL
DIAGRAM
HERE



matasano



symantec™



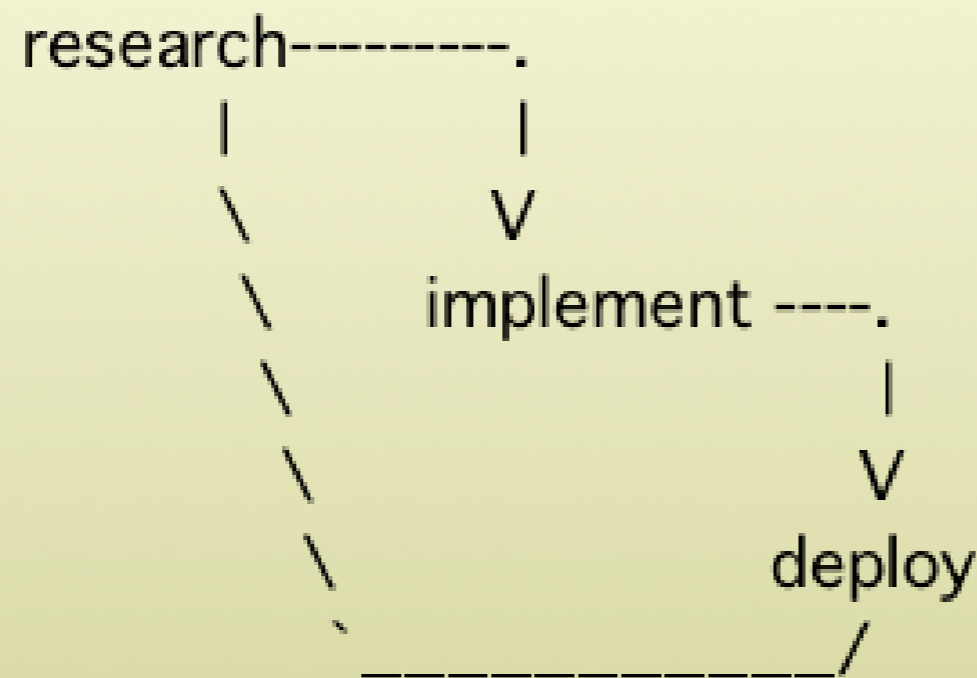
root labs

Conclusion

a cat and mouse game

The Cat And Mouse Process

Diagram: “research”, “implement”,
“deploy”, “repeat” (circles, arrows, huge
cliche, boring!):





matasano



symantec™



root labs

Questions are your
way of proving to
Joanna that
you paid attention.