Don’t Tell Joanna, The Virtualized Rootkit Is Dead

Agenda

- Who we are and what we do
- Virtualization 101
- Vitriol/Hyperjacking (and other HVM Rootkits)
- Why detecting HVMs aren't as difficult as you think
- Pro Forma Punditry
- Q & A

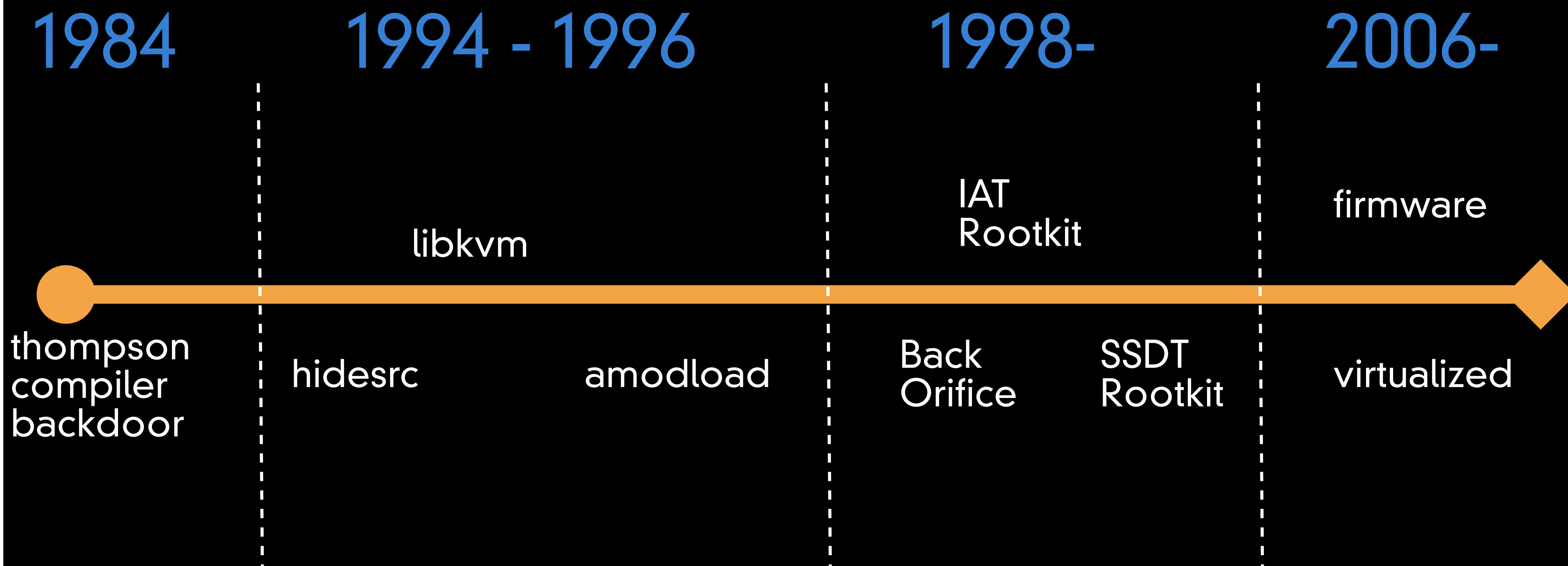
about:nate.lawson

- ★ Co-designer of the Blueray disc content protection lay (at Cryptography Research)
- ★ FreeBSD Committer since 2002
  - Author/maintainer of power management and ACPI kernel code
- ★ Designer of ISS RealSecure NIDS
- ★ Now: independent security consultant (Root Labs)
  - Embedded and PC platform security, crypto design (e.g.: Chumby microcontroller-based authentication)

about:matasano

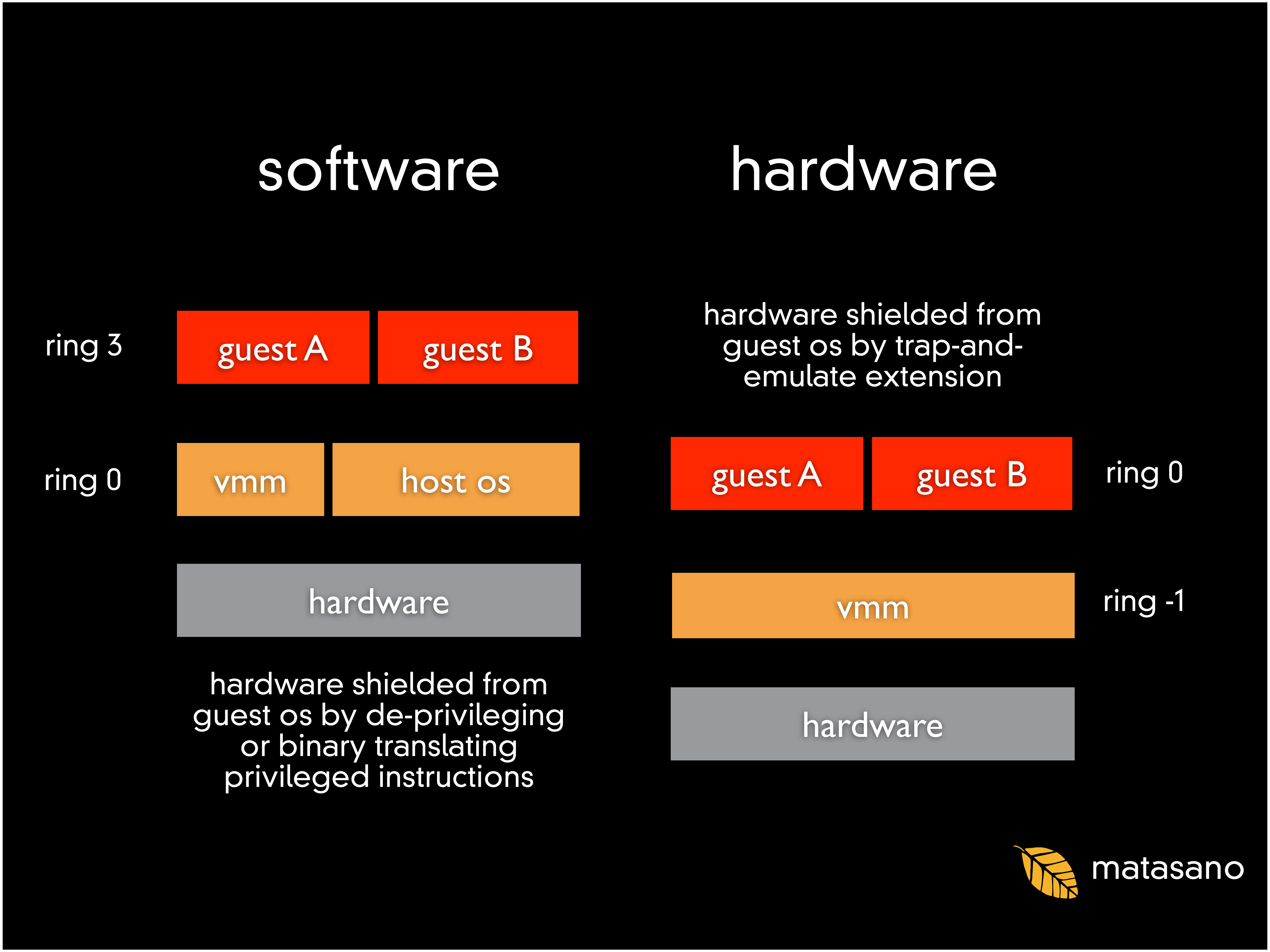
- ★ An Indie Security Firm: Founded Q1’05, Chicago and NYC.
- ★ Research:
  - hardware virtualized rootkits
  - endpoint agent vulnerabilities
  - windows vista (on contract to msft)
  - storage area networks (broke netapp)
  - a protocol debugger
  - 40+ pending advisories

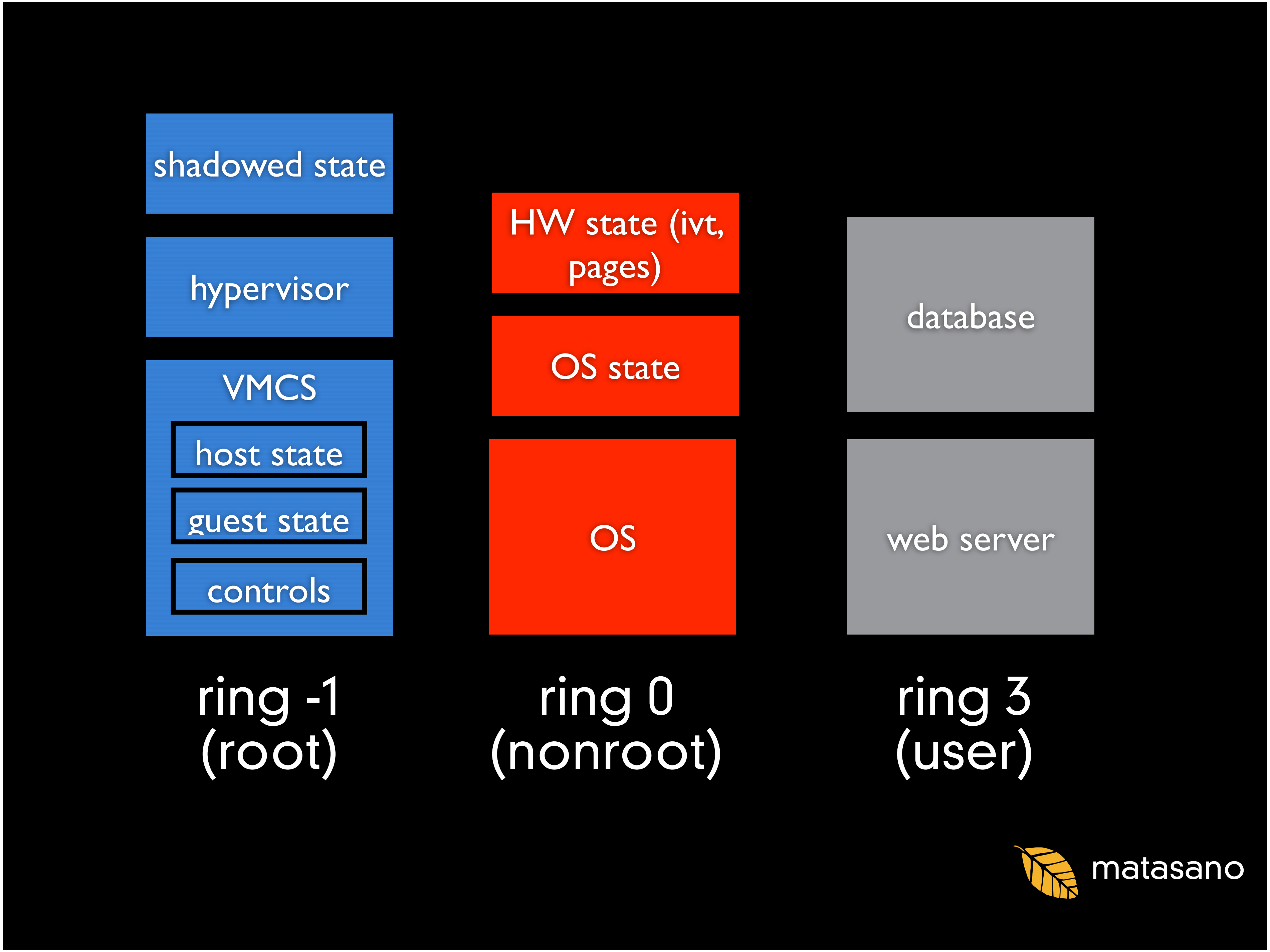
rootkit highlights



lightning intro to VT

matasano


software
hardware
ring 3
guest A
guest B
ring 0
vmm
host os
hardware
hardware shielded from guest os by de-privileging or binary translating privileged instructions
hardware shielded from guest os by trap-and-emulate extension
guest A
guest B
ring 0
vmm
ring -1
hardware
matasano


shadowed state
hypervisor
VMCS
host state
guest state
controls
ring -1
(root)
HW state (ivt, pages)
OS state
OS
ring 0
(nonroot)
database
web server
ring 3
(user)
matasano

insn	purpose
vmxon	enable VT
vmxoff	disable VT
vmclear	initialize VMCS
vmptrld	load current VMCS
vmptrst	store current VMCS
vmread	read values from VMCS
vmwrite	write values to VMCS
vmlaunch	start and enter virtual machine
vmresume	re-enter virtual machine
vmcall	exit virtual machine

sequence of events

- ★ (1) guest OS accesses an msr
- ★ (2) vt traps, looks up host eip
- ★ (3) host calls trap handler
- ★ (4) trap handler emulates msr access
- ★ (5) trap handler incrs guest IP
- ★ (6) trap handler issues vmresume
- ★ (7) guest OS continues

why this is interesting

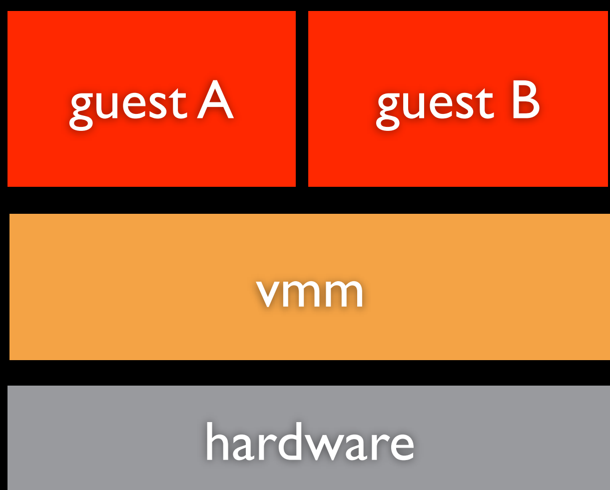
- VT is swapping entire OS-visible state in/out of memory (with API for access)
- Guests have direct device access (unless you prevent them)
- No software bit says “we’re virtualized”.

how we use VT

matasano

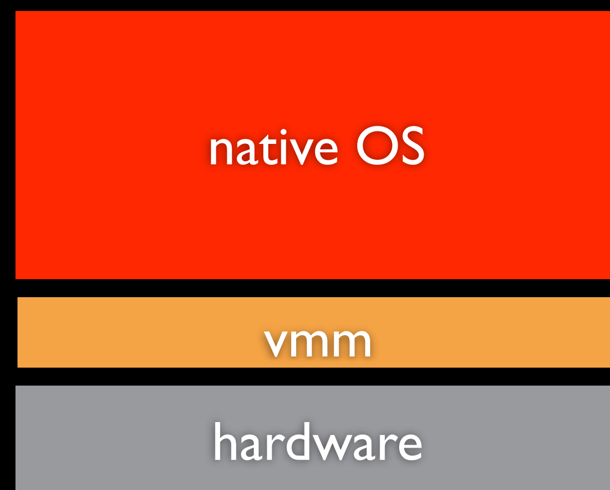
hyperjacking

intended use case



"heavy" vmm runs full-fledged guest machines on servers

rootkit use case



"thin" vmm proxies access to hardware, keeps original OS running

matasano

minimal implementation;
“client” and “server”
do most of the work.

matasano

hyperjacking advantages

- “Impossible to detect” (trap, emulate, and evade detection attempts; MITM the CPU)
- Actually easier than kernel object manipulation
- Potentially OS-independent (portable)
- Potential shellcode payload (fully weaponized)

vitriol: hyperjacking darwin/FreeBSD

- ★ Installed on the fly ("fork" the CPU)
- ★ Hypervisor and guest share CPU state: hypervisor can call into the OS
- ★ (Almost) no shadowed state (just one VM)
- ★ Pass (don't trap) most events.
- ★ Proxy (don't emulate/monitor) most traps.

vitriol: how it works

- (1) get to cpl0
- (2) check cpuid, feature msr for VMX
- (3) allocate vmx and vmcs from IOMalloc
- (4) initialize vmcs, call vmclear
- (5) copy segments, stack, cr3 to vmcs host and guest
- (6) set host(/root/hypervisor) eip to trap handler
- (7) set exec controls to pick events we want
- (8) vmptrld to add vmcs
- (9) (a) vmlaunch (b) vmcall (c) vmresume

Vitriol is less than 1000 lines of code.

matasano

compare to bluepill

- ★ Same concept (hyperjacking proxy vmm)
- ★ Joanna uses AMD SVM
- ★ We don’t support nested VMs
- ★ We don’t hook the network (localhost only)
- ★ We don’t load stealthily (darwin kext)
- ★ Vitriol is a toolkit for detection experiments

HVMs in 2007

- Full Nesting Support
  - *Allow other hypervisors to operate*
- Timing Detection and Submarining
  - *Cat and Mouse Detect / Evade*
  - *Detect Detection and Remove Itself*
- Direct Driver Access
  - *No need to hook the OS*
- Weaponized Hypervisor
  - *HVM as kernel BO payload "shellcode"*

what do we think?

matasano

are hvm rootkits a win?

- SIMPLE
- PORTABLE
- UNDETECTABLE

simple?

- VT is 10 instructions.
- No OS deps in our code
  - except loader and payload
- ~700 lines of boilerplate (expect all hvm rootkits to share)

portable?

- ★ We haven't yet ported to Win32.
- ★ It doesn't look hard.
  - Need to rewrite loader and payload

undetectable?

kernel: fingerprints	*vt: smoking gun*
ssdt/syscall table	
function pointers	
ivt	
hidden pages	hyperjacked vm root
function detours	
hidden threads	
hidden processes	
etc etc etc	

VT-x may be hard to detect.

matasano

VT-x plus a software VMM isn't.

matasano

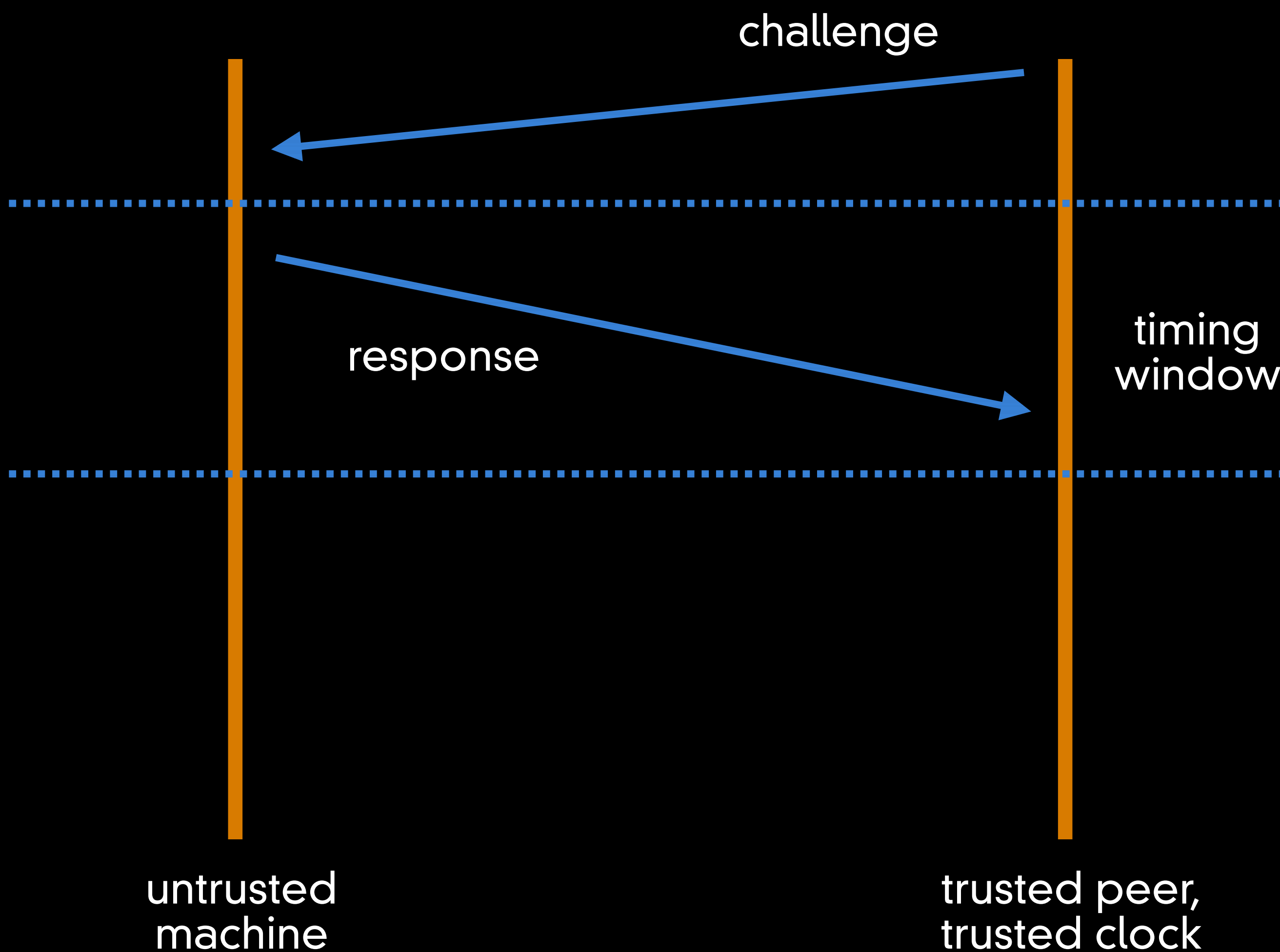
detection heuristics

- ★ FUNCTIONAL: behavior or state changes introduced by hypervisor.
- ★ SIDE-CHANNEL: timing variations introduced by hypervisor.

detection goal

```
int is_virtualized(void);
```

backup goal


challenge
response
timing
window
untrusted
machine
trusted peer,
trusted clock

analog: sniffer detection

- GOAL: Find hacked servers with promiscuous sniffers.
- TARGET: Promisc mode turns off MAC filtering.
- FUNCTIONAL: Target responds to ping with wrong MAC.
- SIDE-CHANNEL: Flood network with nonexistent MAC, measure ping.

measurement strategies

- ★ DIRECT: time an instruction that causes a vm exit.
- ★ INDIRECT: time state (cache, btb) before and after instruction that causes vm exit.

direct measurement

- ★ (1) rdtsc
- ★ (2) cpuid 1,000,000 times
- ★ (3) rdtsc
- ★ if clean: ~200 cycles
- ★ if hyperjacked: ~40,000 cycles

the problem with direct measurement

- Hypervisor controls the TSC!
- (1) on exit: save tsc
- before re-entrance:
  - (2) take delta + exit overhead
  - (3) subtract from TSC offset
- ~5 lines of code. This is a basic feature of VT-x and SVM.

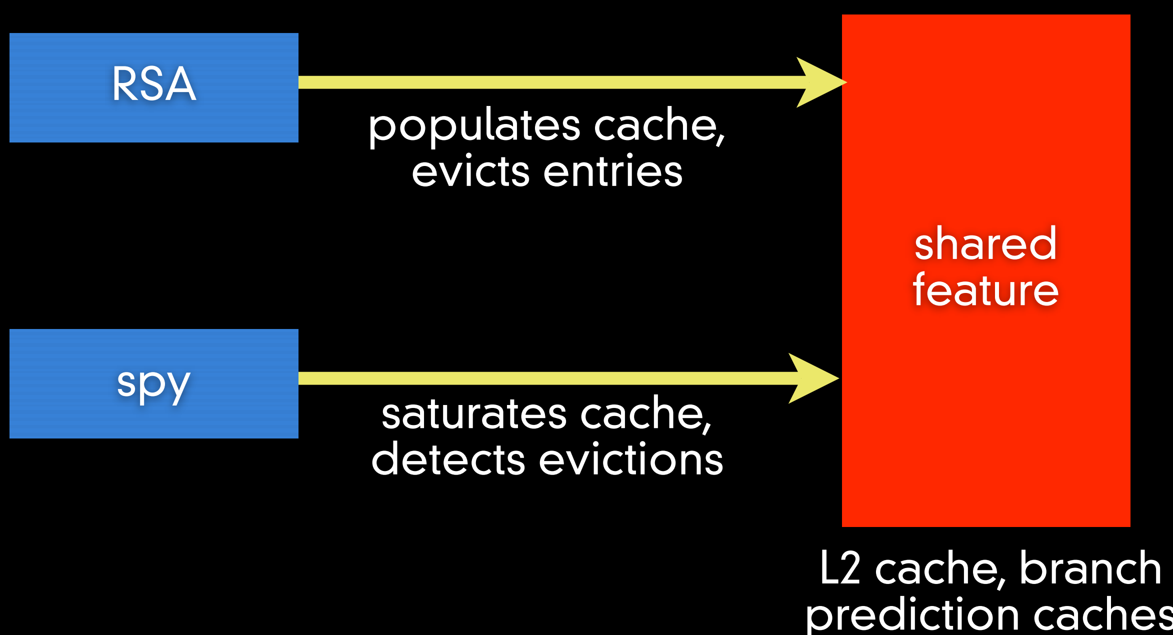
one workaround

- ★ Use counters they didn’t think of:
  - HPET counters
  - Performance counters
  - ACPI timers
  - MSRs that betray timing and latency
- ★ They all need to agree for attackers to win
- ★ But attackers do control all of them

crypto timing attacks

- ★ aciicmez, tromer, bernstein, seifert
- ★ indirect microarchitecture measurement recovers secret crypto keys

cache timing



indirect measurement

- (1) saturate a cache
- (2) baseline cache hits with rdtsc
- (3) cpuid
- (4) repeat baseline
- if clean: (2) and (4) agree
- if hyperjacked: stuff evicted from cache

advantages we have over cryptanalysts

- same cpu, same thread
- not data-independent or oblivious
- extensive shared state
- don’t need to know chinese remainder theorem

conclusions

- How to make life hard for attackers:
  - Introduce data-dependence (many heuristics, not just one)
  - Force them to emulate the microarchitecture (indirect timing of cache, branch buffers)
  - Force them to emulate obscure features (HPET, PerfCounters, AGP GART)
  - Tie them to a single architecture (Intel VT, not Broadcom, Op Roms, etc)

matasanochargen

www.matasano.com/log

matasano