Machine Virtualization:
Efficient Hypervisors, Stealthy Malware

Muli Ben-Yehuda

Technion &
Hypervisor Technologies and Consulting Ltd
Background: x86 machine virtualization

- Running multiple different **unmodified** operating systems
- Each in an isolated virtual machine
- Simultaneously
- On the x86 architecture
- Many uses: live migration, record & replay, testing, . . ., security
- Foundation of IaaS **cloud computing**
- Used **nearly** everywhere
How does it work?

- Popek and Goldberg’s virtualization model [Popek74]: Trap and emulate
- Privileged instructions trap to the hypervisor
- Hypervisor emulates their behavior
- Without hardware support
- With hardware support
What is a rootkit?

- First you take control. How?
- Then you hide to avoid detection and maintain control. How?
- Usual methods are ugly and **intrusive**: easy to detect!
- Can rootkit authors do better?
Hypervisor-level rootkits

- Hypervisors have full control over the hardware
- Hypervisors can trap any operating system event
- Code can enter hypervisor-mode at any time
- **Bluepill**: run the rootkit as the hypervisor
Blue Pill Idea (simplified)

Native Operating System

CALL bluepill

PROC bluepill

enable SVM

prepare VMCB

VMRUN

check VMCB.exitcode

Blue Pill Hypervisor

VMCB

RIP

RET

Native Operating System continues to execute, but inside Virtual Machine this time...

© COSEINC Research, Advanced Malware Labs, 2006
Recursive Bluepill

- Bluepill installs itself on the fly
- Bluepill is now the hypervisor
- Reminder: x86 only supports one hypervisor in hardware
- So how can you bluepill bluepill?
Efficient nested virtualization for Intel x86 based on KVM
Runs multiple guest hypervisors and VMs
What is the Turtles project? (cont’)

- Nested VMX virtualization for nested CPU virtualization
- Multi-dimensional paging for nested MMU virtualization
- Multi-level device assignment for nested I/O virtualization
- Micro-optimizations to make it go fast
Theory of nested CPU virtualization

- Trap and emulate [PopekGoldberg74] ⇒ it’s all about the traps
- Single-level (x86) vs. multi-level (e.g., z/VM)
- Single level ⇒ one hypervisor, many guests
- Turtles approach: $L_0$ multiplexes the hardware between $L_1$ and $L_2$, running both as guests of $L_0$—without either being aware of it
- (Scheme generalized for $n$ levels; Our focus is $n=2$)

![Diagram showing nested CPU virtualization](image-url)
Detecting hypervisor-based rootkits

- Bluepill authors claim “undetectable”
- “Compatibility is Not Transparency: VMM Detection Myths and Realities” [Garfinkel07]
- Hardware discrepancies
- Resource-sharing attacks
- Timing attacks: PCI register access, page-faults on MMIO access, cpuid timing vs. nops
- Can you trust time?
The Dual Role of a Hypervisor
Background: interrupts

- I/O devices raise interrupts
- CPU temporarily stops the currently executing code
- CPU jumps to a pre-specified interrupt handler
Interrupts as an Attack Vector

- Follow the White Rabbit [Rutkowska11]
- Tell the device to generate “interesting” interrupts
- Attack: fool the CPU into SIPI
- Attack: syscall/hypercall injection
- In interrupt-based attacks an untrusted guest generates malicious interrupts which are handled in host mode
- Protect: handle interrupts in guest—not host—mode
- Serve: bare-metal performance!
ELI: direct interrupts for unmodified, untrusted guests
All interrupts are delivered directly to the guest
Host and other guests’ interrupts are bounced back to the host
... without the guest being aware of it
Guests signal interrupt completions by writing to the Local Advance Programmable Interrupt Controller (LAPIC) End-of-Interrupt (EOI) register

Old LAPIC: hypervisor traps load/stores to LAPIC page

x2APIC: hypervisor can trap specific registers

Signaling completion without trapping requires x2APIC

ELI gives the guest direct access only to the EOI register
Threats: malicious guests might try to:

- keep interrupts disabled
- signal invalid completions
- consume other guests or host interrupts
ELI: protection

- **VMX preemption timer** to force exits instead of timer interrupts
- Ignore spurious EOIs
- Protect critical interrupts by:
  - Delivering them to a **non-ELI core** if available
  - Redirecting them as **NMI**s → unconditional exit
  - Use **IDTR limit** to force #GP exits on critical interrupts
Conclusions

- Machine virtualization be used for good, or evil
- How do you protect and serve?
- Happy hacking!
Questions?

muli@cs.technion.ac.il
mulix@hypervisorconsulting.com