Performant Security Hardening

Steve Rutherford
(srutherford@google.com)
Google’s Virtualization Security Team
Preface

This talk is x86 and KVM centric
Outline

Background

Current Hardening

Split Irqchip, PIT

Prototype Hardening

Instruction Emulator
Threat Model

Untrusted Users with **Code Execution in VM**

- E.g. Google Compute Engine

**Guest triggerable bugs** are the biggest concern:

- Guest Triggerable DoS of Host (e.g. x2APIC fallthrough: CVE-2016-4440)
- Guest Escape into Host (e.g. PUSHA emulation: CVE-2014-0049)
- Information Leaks to Guest (e.g. MSR 0x2F8: CVE-2016-3713)

There have been **41 guest triggerable CVEs** since 2009
Historical Security Strategy

Code auditing and fuzz testing

Fairly successful

Found about 15 CVEs in KVM
Goal

Reduce KVM’s **Guest Accessible Attack Surface**

... without impacting performance.
Guest Accessible Attack Surface

Any privileged code with an interface accessible to the guest

Ways of estimating:

Lines of code

Pages of specification

# of Historical CVEs
Attack Surface Reduction

Reduce the amount of guest accessible attack surface

Reduce the privilege of guest accessible attack surface
Why Put Code in Userspace?

**Lower Privilege:** Syscall boundary between userspace and kernel

**Exploit Mitigations and Sandboxing** are more easily deployed in userspace

(ASLR, AppArmor, seccomp-bpf, …)
Why *Not* Put Code in Userspace?

Performance.

*Userspace devices require KVM Exits*

Higher Latency

Lower Throughput
What can be moved to Userspace?

Code that’s complex, slow, and rarely used, but necessary.

Legacy Devices

e.g. PIC, PIT, I/OAPIC

“Edge-Case” Handlers

e.g. Instruction Emulator, MSR handling
Outline

Background

Current Hardening

Split Iqrchip, PIT

Prototype Hardening

Instruction Emulator
What’s the Irqchip?

KVM uses the term irqchip to refer to the interrupt controllers

i.e. the PIC, I/OAPIC and APIC for x86

KVM has supports both userspace and kernel irqchips

The kernel irqchip provides a significant perf boost over userspace irqchip
Split Irqchip

Take the best of Userspace and Kernel Irqchips

The **PIC** and **I/OAPIC** aren’t used often by **modern VMs**…

… but the **APIC** is.

So move the **PIC** and **I/OAPIC up to userspace**, and add necessary API to communicate between userspace and the in-kernel APIC
Programmable Interrupt Controller

Interrupt controller that maps directly from GSI to interrupt

Can’t live without it

Necessary for **real mode interrupts** (16-bit) during early boot

Allows legacy devices like the RTC and the PIT to send interrupts

Masked early in boot and **replaced by the I/OAPIC**
Why worry about the PIC?

Medium attack surface

~1% of x86 KVM (by LoC), 24 page spec

Complex API

Tons of modes: AutoEOI vs EOI, Auto vs specific rotate…

Non-trivial amount of unspecified behavior
Why not keep the PIC in KVM?

Rarely Used

Indefinitely \textit{masked during boot} of every common OS

Slow

Already \textit{requires VMEXITs} and instruction emulation
Necessary Interface

Send **local interrupts from userspace**

Updated existing `kvm_vcpu ioctl KVM_INTERRUPT`

Added support for userspace **interrupt windows** with in-kernel APIC

Hijack fields normally used by userspace irqchip
I/OAPIC

Global Interrupt Controller: Configurable mapping from level and edge-triggered GSIs to APIC interrupts.

T.L.D.R.: PIC for Multicore

Necessary for any non-MSI-supporting (INTx) device (PIT, RTC, …)

Devices using MSIs bypass the I/OAPIC
Why worry about the I/OAPIC?

CVE-2013-1798

Arbitrary Read “Guarded” by an Assert

CVE-2014-0155

Denial of Service via invalid Redirect Table Entry

Complexity

>1% of x86 KVM (810 LoC), 20 page spec
Why not keep the I/OAPIC in KVM?

Performance degradation for INTx

Level-triggered EOI{s} now go to userspace

Performance degradation for I/OAPIC reads and writes

VMEXITs for the I/OAPIC are now also KVM_EXITs

… But performance sensitive devices are likely using MSI/MSIx already
Necessary Interface

Send Interrupts to APICs

Available via kernel GSI routing table and kvm_set_irq

Added **EOI KVM exit**

Made the **EOI Exit Bitmap** configurable by userspace

Via configurable range in GSI routing table
Programmable Interval Timer

- Fixed frequency timer with multiple counters
- Commonly used to calibrate other timers/counters
Why worry about the PIT?

CVE-2015-3214

PIT Out of Bound memory access

Other CVEs

3x Denial of Service

Complexity

>1% of x86 KVM (804 LoC), 21 page spec
Why not keep it in the kernel?

Rarely Used

Typically only used during boot

Slow

Reads and writes already require VMEXITs and instruction emulation
Current Hardening Results

Moves 4-5% of KVM into userspace and the sources of 6/41 guest triggerable CVEs since 2009

Negligible performance impact for Modern VMs
Disk Random Read kIOPs (Modern/MSIx)
How Modern is a Modern VM?

Performance critical devices (e.g. disk, network) need **MSI support**

Necessary to **skip the I/OAPIC**

Guest should not read/write to I/OAPIC or PIT frequently
Disk Random Read kIOPs (**Legacy/INTx**)

![Bar chart showing Disk Random Read kIOPs for Ivybridge and Haswell processors with two categories: Kernel Iirqchip and Split Iirqchip.]
Try It Out

**Split Irqchip** and **userspace PIT** supported by **x86 QEMU v2.6** and up via `-machine kernel_irqchip=split`

Needs $\geq$ **4.4 Linux Kernel**
Outline

Background

Current Hardening

Split Irqchip, PIT

Prototype Hardening

Instruction Emulator
Instruction Emulator

Emulates x86 instructions with x86, when the CPU can’t virtualize.

Most commonly reads and writes to emulated I/O
Why worry about the Instruction Emulator?

Converts *arbitrary bytes* into an instruction…

… then *emulates that instruction*

Tons of Instructions

- PUSHA, POP, SYSCALL, MOV, CMPXCHG8B…

5500 Lines of Code (and growing)

11 CVEs since 2009
Why not keep the emulator in the kernel?

With **Split Irqchip** almost all **MMIO devices are now in userspace**

APIC is an exception, but **APICv skips emulation**
Userspace Instruction Emulation Interface

Needs guest register access

Already available, but could be accelerated using kvm_sync_regs

New KVM exit type

KVM_EXIT_EMULATION_NEEDED

Must be able to communicate with any kernel MMIO/PIO device

Need ability to read/write to APIC => new IOCTL
Disk Random Read kIOPs (Modern/MSIx)
How Modern is a Modern VM?

Host CPU needs APICv (>= Ivybridge)

For APIC Access VMEXITs

In addition to Split Irqchip Requirements
Disk Random Read kIOPs (Legacy/INTx)
Attack Surface Reduction Summary

Removed the sources of \textbf{17/41} the Guest Triggerable CVES

\textbf{Removed} \textbf{>7500 LoC} from KVM (\textasciitilde15\% of x86 KVM)

\textbf{Negligible perf impact} for modern VMs
Future Work

Finalize *interface for Userspace Instruction Emulation*

Add *Userspace Instruction Emulation* Support to *QEMU*

Add *Userspace MSR Handling* to *KVM* and *QEMU*

Test *Performance on QEMU*

(Prior performance testing done with Google’s userspace VMM)
Questions?