Evaluate implementation options of KVM-based Type1 (or 1.5) hypervisor

Jun Nakajima

Contributors: Chuanxiao Dong, Anthony Xu
Legal Disclaimers

• Intel provides these materials as-is, with no express or implied warranties.
• All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice.
• Intel processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.
• Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at http://intel.com.
• Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.
• Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.
• *Other names and brands may be claimed as the property of others.
© Intel Corporation 2020
Agenda

• Motivation
• Implementation Options
• PoC
• Performance Data
• Our Conclusion
• Next Step
Security Risks of Linux/KVM Guests

- KVM piggybacks on Linux
  - More attack surfaces, making guests more exposed...
- Full access by user-space VMM
- Full access by KVM/Linux Kernel
  - To any guest VM memory, vCPU states, etc.

*: From presentation last KVM Forum: “Manage Session Enhancing KVM for Guest Protection and Security”
Motivation of Type 1.5 Hypervisor

• Separate Hypervisor functionality from Linux
  – Linux handles I/O and user processes
  – Hypervisor is responsible for isolation
    • Thus needs to be trusted
• If trusted, hypervisor can create secure environment
  – TEE (Trusted Execution Environment)
  – Trusted VMs
Converting KVM to Type 1.5

Maintain: I/O passthrough

Domain 0 (Dom0)

Input/Output

Hypervisor (L0)

H/W

User-level KVM VMs

H/W

Linux Kernel

KVM

VMs

User-level

VMs (L2)

VMs (L1)

VMs (L1)

Linux Kernel

H/W
Two Extremes

Dom0

User-level

Linux Kernel (L1)

KVM

VMs (L2)

User-level VMM for Dom0

User-level VMM for Dom0

OS functionality (scheduler, memory mgmt, etc.)

Deprivilege Linux for isolation

Mini-conf Linux/KVM

Reactivate (no scheduler)

Boots first and separately

Can be loaded by Linux
**Linux/KVM Hypervisor**

**Pros**
- Unmodified guests on L1
- Benefits from Linux/KVM

**Cons**
- Higher latency to Dom0
  - Scheduling, VM exits
- Still big (e.g. TCB)
  - Maybe we can deal with it…
- Virtio for guests
- Power management (PM)?
  - Who should manage power for CPUs and platform
Dom0: Scheduling and PM Issues

- Hypervisor needs to own VM scheduling
  - Intercept `HLT/MWAIT` in Dom0
- Inefficient for clients:
  - Two-level scheduling
    - VM-level and process-level (within VM)
  - Unexpected latencies in VMs, especially Dom0
Impacts of Linux/KVM Hypervisor

• How to create VMs?
  – Need to invoke QEMU process on the host from Dom0 (a guest)
    • E.g. Nitro Enclaves driver
• Virtio
  – No I/O devices are available in hypervisor
  – Only memory filesystem
Lightweight Hypervisor

- **Pros**
  - Same code path of bare-metal Linux/KVM
  - Low latency & overhead
    - No VM exits if dom0 behaves legitimately
  - Small TCB

- **Cons**
  - Limited L1 VM types
    - E.g. no virtual devices support
  - L2 for Unmodified guests
    - Overhead compared with L1?
Optimization for KVM Guests

- Optimized nested virtualization using VMCS shadowing
  - Passthrough shadow VMCS (for most fields) in L1
  - Convert shadow VMCS to real VMCS quickly (flip one bit)
- KVM 1st level Entry Point
  - Fast VM entry/exit if exit handling doesn’t require Linux services
  - Allow KVM VMs to run as L1
Optimized Nested Virtualization

Current Implementation

Optimized L2 VM exit/entry

*: Bit 31: shadow-VMCS indicator in VMCS region

Copy/Sync

Flip the bit*
PoC: Lightweight Hypervisor by extending VBH

- Original VBH* (Virtualization Based Hardening)
  - Deprivileges Linux kernel to harden the kernel (Dom0)
    - With all I/O and APIC passthrough
- Added simple nested virtualization to run KVM guests (L2)
  - Only for L1 VM (bare-metal VM, where GPA = HPA)
  - Implemented optimized VMCS shadowing, virtual EPT for isolation
- Added a feature to run a simple L1 VM in TEE
  - E.g. OP-TEE OS**
- Working on virtual IOMMU

*: From presentation KVM Forum 2019:
“Manage Session Virtualization Based Hardening: Securing Container Workloads and Beyond”

**: https://github.com/OP-TEE/optee_os
Comparing Performance 1/2

Linux Kernel (L1)

KVM

User-level

VMs (L2)

Dom0

Lightweight Hypervisor (L0)

LH L2

User-level

VMs (L2)

KVM

Linux Kernel (L1)

Hypervisor

User-level VMM for Dom0
e.g. QEMU

KVM (L0)
## KVM L2 and LH L2

Kernel Build L2 VM Exit Breakdown

<table>
<thead>
<tr>
<th></th>
<th>Hypervisor</th>
<th>VM Exit: L2→L1</th>
<th>L1 Handler</th>
<th>VM Entry: L1→L2</th>
<th>Total</th>
<th>Improvement Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>External Interrupt</strong></td>
<td>Linux/KVM</td>
<td>27172</td>
<td>147345</td>
<td>12058</td>
<td>186575</td>
<td>Total Improvement:40%</td>
</tr>
<tr>
<td></td>
<td>LH</td>
<td>2418</td>
<td>108332</td>
<td>1087</td>
<td>111838</td>
<td>L1→L2 Switch:47% Handler:53%</td>
</tr>
<tr>
<td><strong>IO_INSTRUCTION</strong></td>
<td>Linux/KVM</td>
<td>17362</td>
<td>483317</td>
<td>26788</td>
<td>526570</td>
<td>Total Improvement:87%</td>
</tr>
<tr>
<td></td>
<td>LH</td>
<td>540</td>
<td>63745</td>
<td>925</td>
<td>65211</td>
<td>L1→L2 Switch:9% Handler:91%</td>
</tr>
<tr>
<td><strong>MSR WRITE</strong></td>
<td>Linux/KVM</td>
<td>18175</td>
<td>67198</td>
<td>17674</td>
<td>103047</td>
<td>Total Improvement:77%</td>
</tr>
<tr>
<td></td>
<td>LH</td>
<td>956</td>
<td>21358</td>
<td>721</td>
<td>23036</td>
<td>L1→L2 Switch:43% Handler:57%</td>
</tr>
<tr>
<td><strong>PREEMPTION_TIMER</strong></td>
<td>Linux/KVM</td>
<td>46058</td>
<td>215206</td>
<td>27750</td>
<td>289015</td>
<td>Total Improvement:91%</td>
</tr>
<tr>
<td></td>
<td>LH</td>
<td>1768</td>
<td>27610</td>
<td>744</td>
<td>30123</td>
<td>L1→L2 Switch:28% Handler:72%</td>
</tr>
</tbody>
</table>

Improvements from flipping shadow-VMCS indicator
Comparing Performance 2/2

- **Dom0**
  - User-level
  - Linux Kernel (L1)
  - KVM
- **LH L2**
  - User-level
  - Linux Kernel (L1)
  - KVM
- **VMs (L2)**
  - User-level VMM for Dom0: e.g. QEMU
  - Hypervisor: Lightweight Hypervisor (L0)

**KVM L1**

- VMs (L1)
- VMs (L2)
Comparing KVM L1 and LH L2 Guest
(without KVM 1st Level Entry Point)

LH L2 and KVM L1 is almost equivalent

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>KVM L1</th>
<th>LH L2</th>
<th>LH L2 vs. KVM L1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel compiling</td>
<td>348.3</td>
<td>353</td>
<td>98.67%</td>
</tr>
<tr>
<td>Unit: second</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iperf</td>
<td>41.2</td>
<td>37.16</td>
<td>90.19%</td>
</tr>
<tr>
<td>Unit: Gb/sec</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Between VM and underlining VMM)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIO seq read</td>
<td>515.8</td>
<td>471.2</td>
<td>91.35%</td>
</tr>
<tr>
<td>Unit: MB/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIO seq write</td>
<td>279.2</td>
<td>232.4</td>
<td>83.24%</td>
</tr>
<tr>
<td>Unit: MB/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIO rand read</td>
<td>256.8</td>
<td>226.6</td>
<td>88.24%</td>
</tr>
<tr>
<td>Unit: MB/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FIO rand write</td>
<td>219</td>
<td>182</td>
<td>83.11%</td>
</tr>
<tr>
<td>Unit: MB/s</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sysbench CPU</td>
<td>4623.66</td>
<td>4609.03</td>
<td>99.68%</td>
</tr>
<tr>
<td>Unit: events per second</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sysbench CPU</td>
<td>8218.38</td>
<td>8207.89</td>
<td>99.87%</td>
</tr>
<tr>
<td>Unit: MiB/sec</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Findings from PoCs

- Linux/KVM Hypervisor has structural impacts:
  - Large structural changes to resource management
    - Scheduling, power management, VM management
  - Virtio implementation
  - It would require different efforts to optimize/tune
    - Beyond current Linux/KVM

- Lightweight Hypervisor
  - LH L2 and KVM L1 is almost equivalent
    - I/O needs more optimization
Our conclusion

• Lightweight (reactive) Hypervisor approach is more suitable for the current Linux/KVM to make it more secure (Type 1.5 VMM)
  – Same code path as bare-metal Linux/KVM, including scheduling and power management, etc.
  – Low latency & overhead
• VBH-based Hypervisor can harden Dom0 kernel and guests additionally
• KVM guests run with minimal overhead
• Advantage when implementing TEE because of small TCB
Next Step

- Finish VBH-based PoC
  - Complete IOMMU virtualization
    - For direct I/O support in secure environment
  - Optimize KVM guest performance more
    - I/O performance (e.g. write operations)
    - KVM 1st Level Entry Point in VBH

- Share the code
  - github