# Enhancing KVM for Guest Protection and Security

Jun Nakajima, Intel Corp.

# Intel Notice & Disclaimer

# Agenda

- Security Implications of KVM
- “Secure Virtualization” for KVM
- Degree of Guest VMs Protection
- Proof of Concept
- Next Steps

# Security Implications of KVM

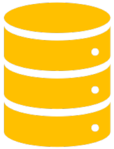
**KVM piggybacks on Linux systems**

More attack surfaces, making guest more exposed

Guests could exploit host via user-space VMM

**Full access by user-space VMM:**

Guest VM memory, vCPU states, etc.



**Full access by KVM/Linux Kernel:**

Any guest VM memory, vCPU states, etc.

# Secure Virtualization for KVM

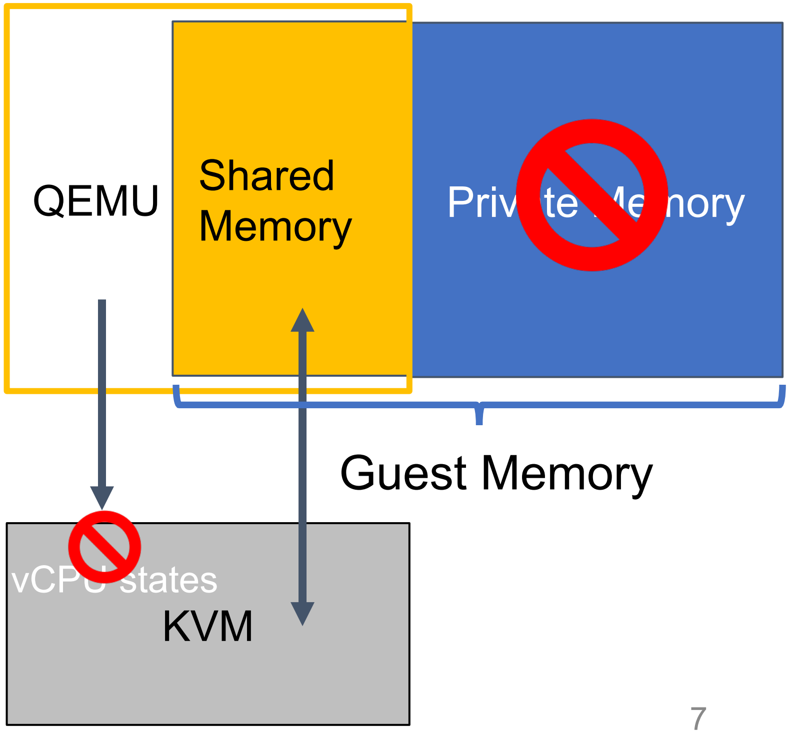
- Inspired by memory encryption technologies
- Protect guest VMs from a benign but vulnerable VMM
- Degree of "Protection" depends on what can be removed from TCB:
  - User-space VMM
  - Misc. kernel-space subsystems/drivers
  - Hypervisor

# Degree of Guest VMs Protection — what to remove from TCB

| To protect potential attacks from | What's required |
|---|---|
| User-space VMM | Deny access to guest register state<br>Deny access to guest private memory |
| Kernel-space (except KVM) + above | Remove direct mapping<br>Deny access to guest private memory |
| Hypervisor (KVM) + above | Hardware-based security feature |

# Secure Virtualization for KVM

- Guest specifies shared regions out of private memory
  - QEMU can access only shared
  - KVM may access private if in TCB
- KVM denies access to guest vCPU state by user-space VMM (e.g. QEMU)



# Modifications to Guest VM (Linux) when KVM is in TCB

- PV operations
  - Controls shared guest memory (GPA)
    - All private at boot time
  - Use swiotlb bounce buffer to force DMA operations in shared memory
  - Etc.
- (Optional) Hypervisor denies access by other kernel subsystems/drivers to guest private memory

# Modifications to Guest VM (Linux) when KVM is not in TCB

- Additional PV operations
  - Avoid VM exiting operations that would require KVM to access guest VM memory
    - E.g. MMIO operations (replace them with Hypercall)
  - Or, handle them in guest VM via virtual exceptions
    - E.g. #VE (Virtual Exception) and emulate operations inside VM (and Hypercall)
- Useful even if KVM is in TCB

# Proof Of Concept

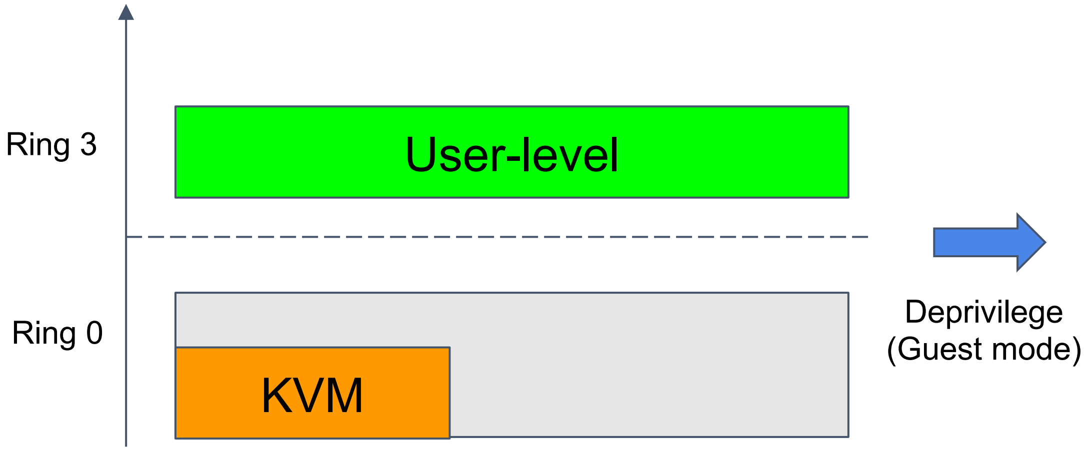
- KVM and other changes
  - Remove mapping from QEMU and kernel
  - “Ideal/optimal memory management for future VMs”, *Isaku Yamahata*, November 1 (11:30 - 12:00)
- Average CPU% overhead (virtio)*:
  - 1.2% (1 VM), 1.3% (10+ VMs) for disk read, 1.1% (1 VM), 1.2% (10+ VMs) for disk write
  - 2.6% for network send (1 VM), 3.8% (10+ VMs)

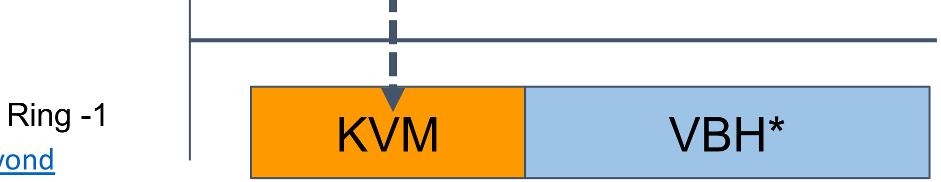
*:

- Xeon Platinum 6140, Skylake processors @ 2.3GHz
- 2 sockets x 18 cores each
- HT OFF, P States OFF, Turbo OFF, C states OFF

# Deny Access to Guest Private Memory (Kernel)

Use EPT to isolate/protect memory



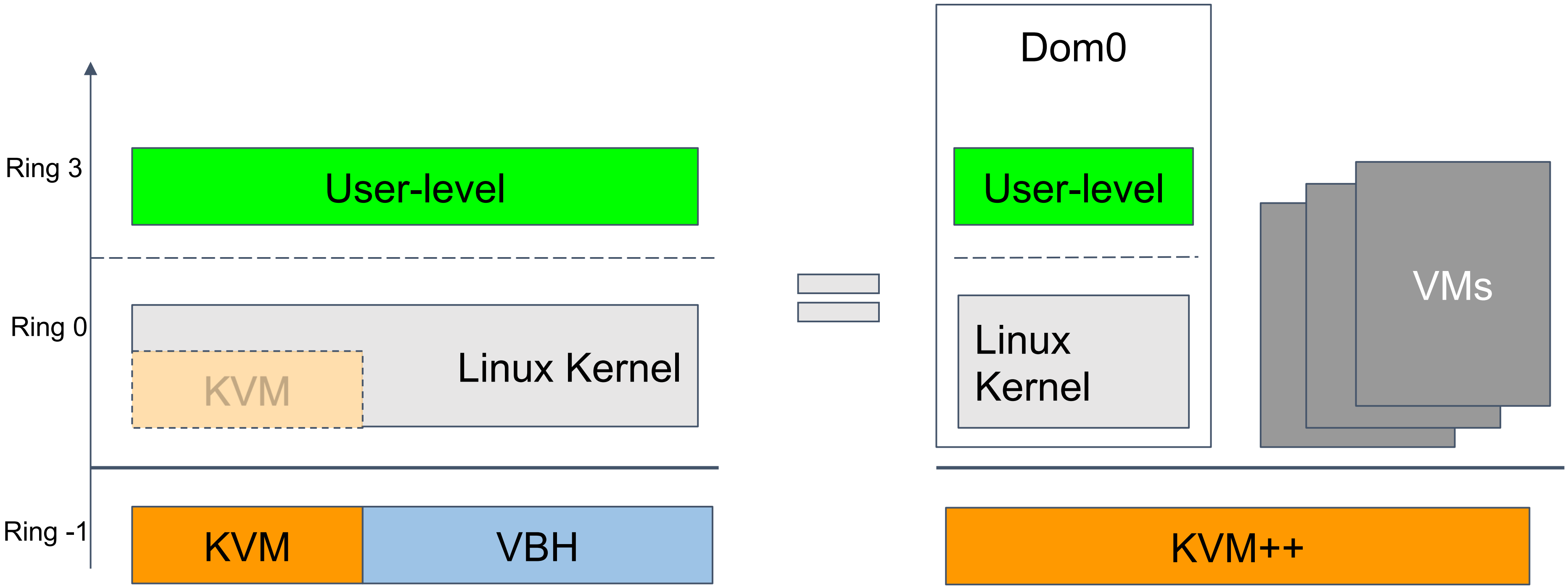




*VBH: Virtualization Based Hardening

**Thursday**, October 31 • 14:45 - 15:15

[Virtualization Based Hardening: Securing Container Workloads and Beyond - Jun Nakajima, Intel & Andrei Lutas, Bitdefender]

# Making Type-1 Hypervisor From KVM



# Next Steps

- Complete PoC and share the patches
  - Changes to Linux, KVM, etc.
    - Linux gust changes: use existing code for AMD SEV as much as possible
- Propose how to remove guest memory mapping from user-space
  - In Backups
- PoC of “Making Type-1 Hypervisor From KVM”

# Backups

# Remove User-space VMM from TCB

- Deny access to guest register state:
  - Reject ioctls() that get/set guest state
  - Other changes required to hide guest state
- Deny access to guest private memory (example):
  - Add new flag, e.g. VM_PRIVATE or so, to vm_flags
  - Kernel removes PTEs for VM_PRIVATE memory from user-space page tables
  - Guest controls shared vs. private GPAs
  - KVM returns -EFAULT to user-space on private access to memory without VM_PRIVATE