

Trust, Confidentiality, and Hardening

The VIRTIO Lessons

Michael S. Tsirkin
mst@redhat.com

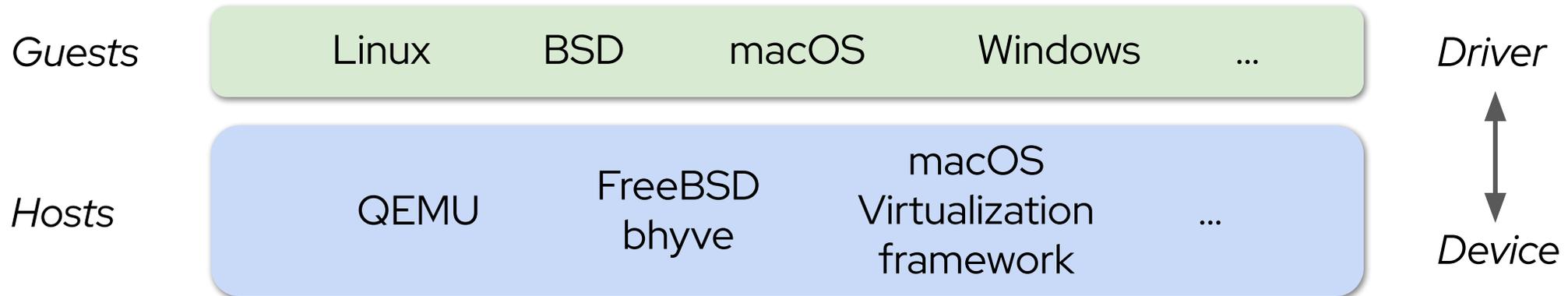
Stefan Hajnoczi
stefanha@redhat.com

Agenda

- ▶ Status of Linux VIRTIO driver trust model
- ▶ How we got here
 - Hypervisor architecture
 - IOMMUs and hardware VIRTIO devices
 - Linux VDUSE
- ▶ Where we are going
 - Confidential VMs with untrusted & trusted (TDISP) devices

VIRTIO

Open standard for I/O devices with broad support across hypervisors and operating systems.



VIRTIO 1.2 specification:

<https://docs.oasis-open.org/virtio/virtio/v1.2/virtio-v1.2.html>

VIRTIO Device Types

Standard device types include:

- ▶ Net - Network interface
- ▶ Blk - Block device
- ▶ GPU - Graphics
- ▶ Serial - Serial console device

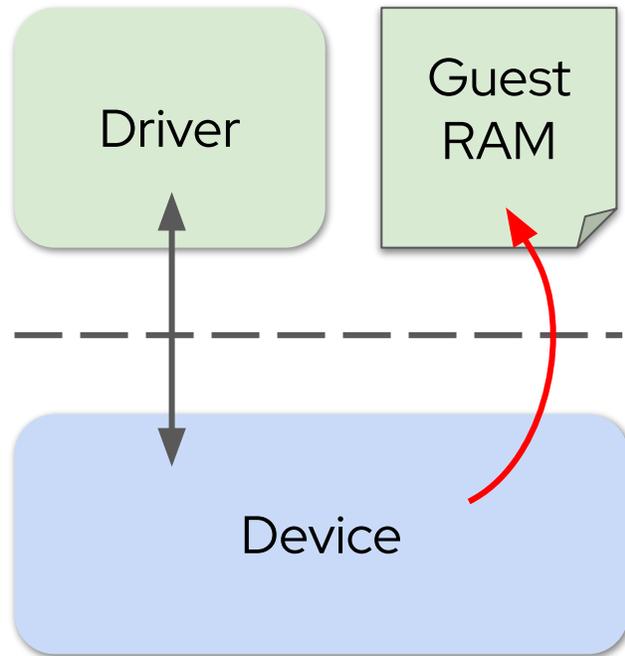
19 standard device types as of VIRTIO 1.2

Applications of VIRTIO

1. Traditional virtualization use case
 - Software paravirtualized devices in hypervisors
2. Linux VDUSE
 - Software devices for bare metal or containers
3. Hardware VIRTIO devices
4. Confidential computing with software or hardware devices
 - Lots of interest from CPU and cloud vendors to offer this

Let's explore their different trust models...

Traditional Virtualization Use Case



Hypervisor and devices have full access to guest RAM

Devices reach into guest RAM to transfer I/O buffers

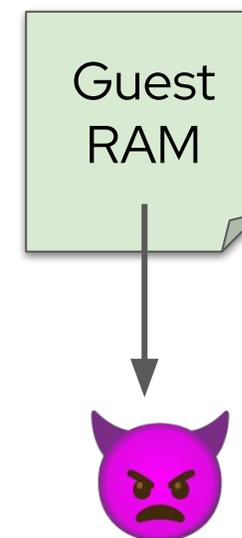
What are the implications?

Malicious Read from RAM

Extract sensitive information (passwords, cryptographic keys, confidential data)

“Can my hosting provider see everything inside my Virtual **Private** Server?”

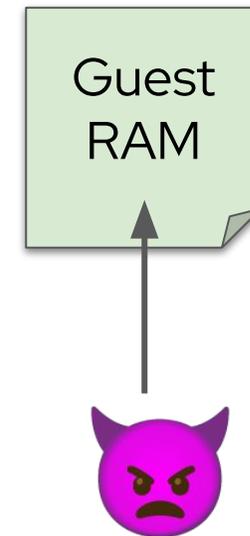
▶ Yes 😁



Malicious Write to RAM

- ▶ Crashing driver/OS/application
- ▶ Changing critical state (process uid/gid)
- ▶ Code execution

The guest is completely controlled by the host



Why Drivers trust Devices

Drivers may trust hypervisor because hypervisor already as full access to guest

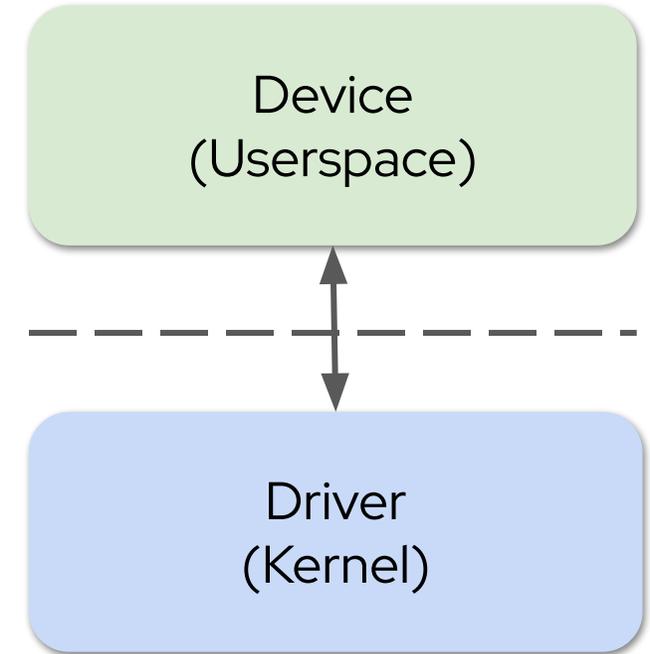
It's futile for drivers to distrust devices built into the hypervisor, too

But VIRTIO use cases expanded and the status quo was no longer acceptable...

Linux VDUSE Use Case

A way to use the same userspace VIRTIO device implementation for guests and bare metal (containers)

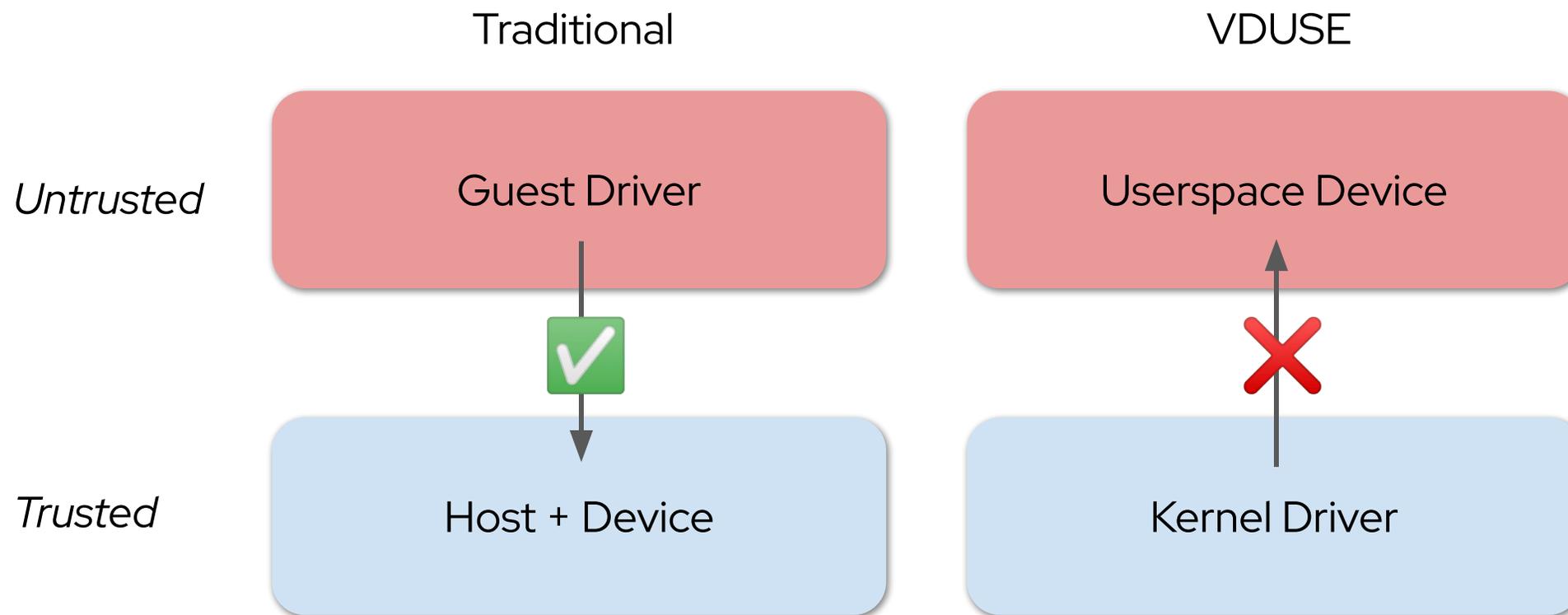
Existing kernel virtio driver now talks to a userspace device instead of a hypervisor



VDUSE overview:

<https://www.redhat.com/en/blog/introducing-vduse-software-defined-datapath-virtio>

Comparing Traditional and VDUSE Trust Models



VDUSE kernel driver must not trust userspace device

Kernel/Userspace Trust Model

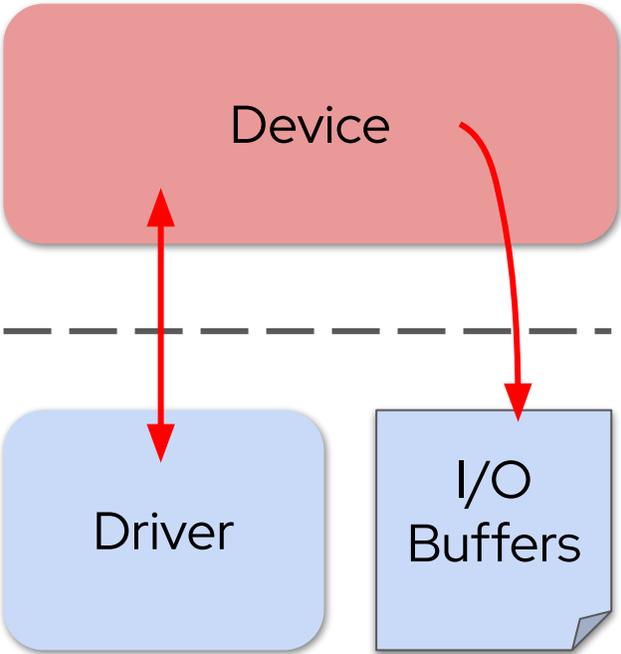
Privileged userspace processes can reconfigure system and access data

BUT

must not corrupt kernel memory or gain kernel code execution

Why? Signed kernel modules, kernel_lockdown(7)

Mistrust in Devices and Drivers



Driver must validate data from device

Driver must isolate device to protect I/O buffers (RAM)

Input Validation

```
commit 6ae6ff6f6e7d2f304a12a53af8298e4f16ad633e
Author: Jason Wang <jasowang@redhat.com>
Date: Tue Oct 19 15:01:43 2021 +0800

    virtio-blk: validate num_queues during probe
...
diff --git a/drivers/block/virtio_blk.c
b/drivers/block/virtio_blk.c
index a33fe0743672..dbcf2a7e4a00 100644
--- a/drivers/block/virtio_blk.c
+++ b/drivers/block/virtio_blk.c
@@ -571,6 +571,10 @@ static int init_vq(struct virtio_blk
*vblk)
                                &num_vqs);
    if (err)
        num_vqs = 1;
+   if (!err && !num_vqs) {
+       dev_err(&vdev->dev, "MQ advertisted but zero
queues reported\n");
+       return -EINVAL;
+   }
```

Linux VIRTIO drivers were checked to add input validation

Multi-person effort

VIRTIO core and many devices now validate all input

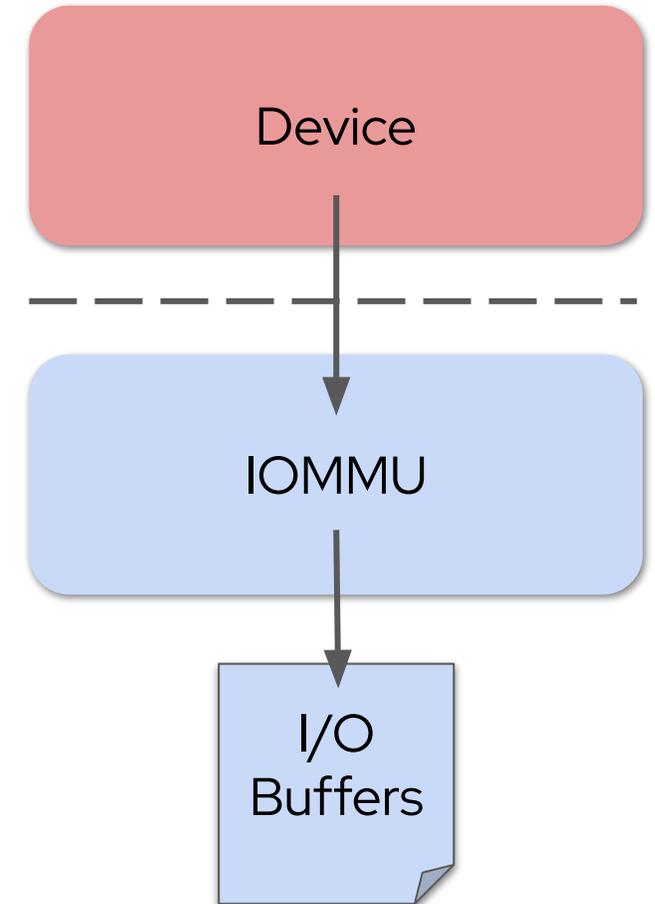
Restricting DMA using an IOMMU

Devices with unrestricted DMA can snoop or corrupt RAM

IOMMU only allows DMA to/from RAM granted by driver

VDUSE has an IOTLB mechanism similar to an IOMMU

- ▶ `ioctl(VDUSE_IOTLB_GET_FD)`



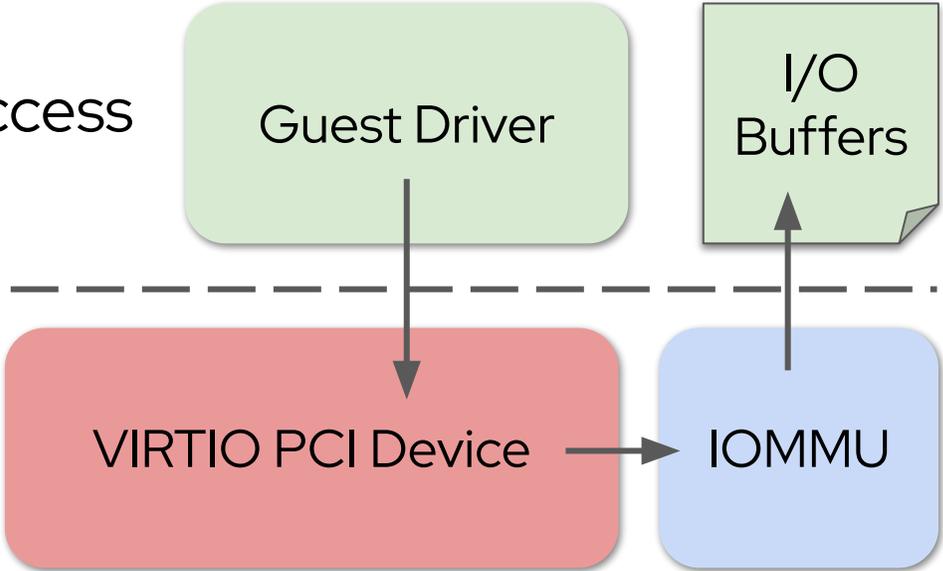
Hardware VIRTIO Devices

Physical PCI adapters available from hardware vendors & custom DPUs

PCI device assignment to guest

IOMMU can be used to restrict device's RAM access

Similar trust model to Linux VDUSE

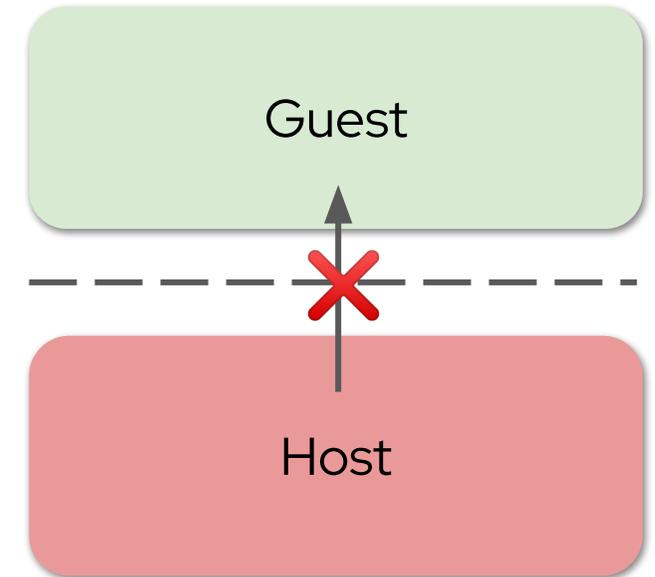


Confidential Computing

Deploy a workload without trusting hosting provider

Host compromise should not compromise guests

Hardware and software still a work in progress



Details on platform specifics:

<https://www.redhat.com/en/blog/confidential-computing-platform-specific-details>

How Confidential VMs Work

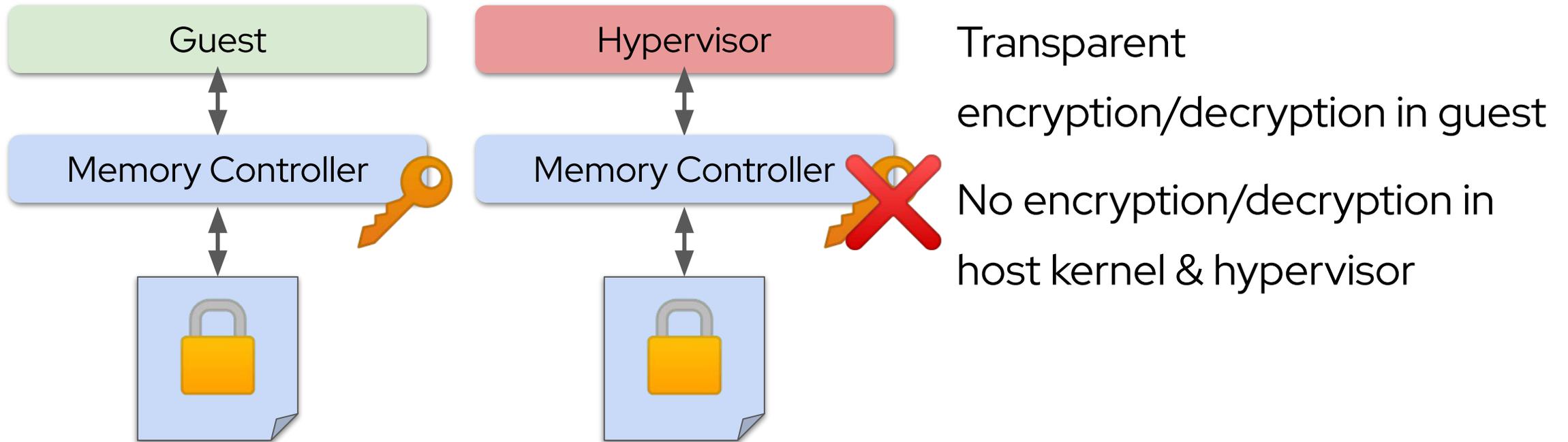
Reduces the Trusted Computing Base (TCB)

- ▶ Host kernel and/or hypervisor no longer trusted
- ▶ CPU still trusted
- ▶ Physical & side-channel attacks, denial of service out of scope

Hypervisor and host kernel cannot access guest:

- ▶ CPU registers are protected
- ▶ Guest RAM is protected

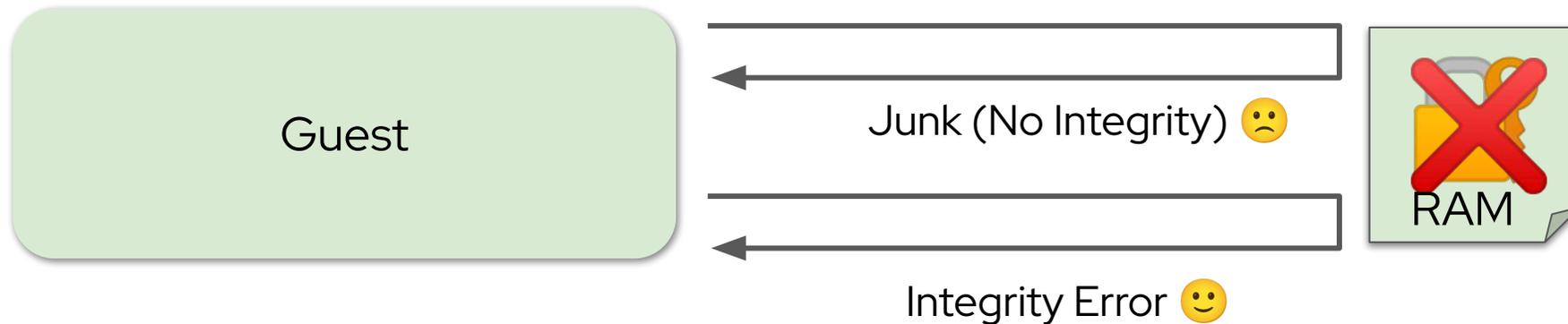
Memory Encryption



Memory Encryption - Integrity Protection

State of the art is encrypted guest RAM with integrity protection (tampering detection)

Without integrity protection, hypervisor or device writes to encrypted memory result in garbage being silently read by driver



Toy Example

Toy example does not work like Intel TDX, AMD SEV, etc (they involve remote attestation) but resembles IBM Secure Execution. Bare minimum to show confidential computing is possible:

1. User provides initial VM memory state encrypted with CPU's public key
2. CPU decrypts initial VM memory state with its private key and places it into encrypted guest RAM
3. Hypervisor launches VM but cannot snoop/tamper with encrypted guest RAM or CPU registers
4. User knows VM is running on a trusted CPU because the image was decrypted with the private key that no one else has

Untrusted Devices in Confidential Computing

Can Confidential VMs use untrusted devices? Yes, some.

- ▶ Network interface - Encrypt communication with (D)TLS so device cannot inspect or tamper with messages
- ▶ Storage device - Encrypt disk with dm-crypt + dm-integrity so device cannot inspect or tamper with stored data

Isolating Untrusted Devices with swiotlb

Linux swiotlb bounce buffers DMA transfers

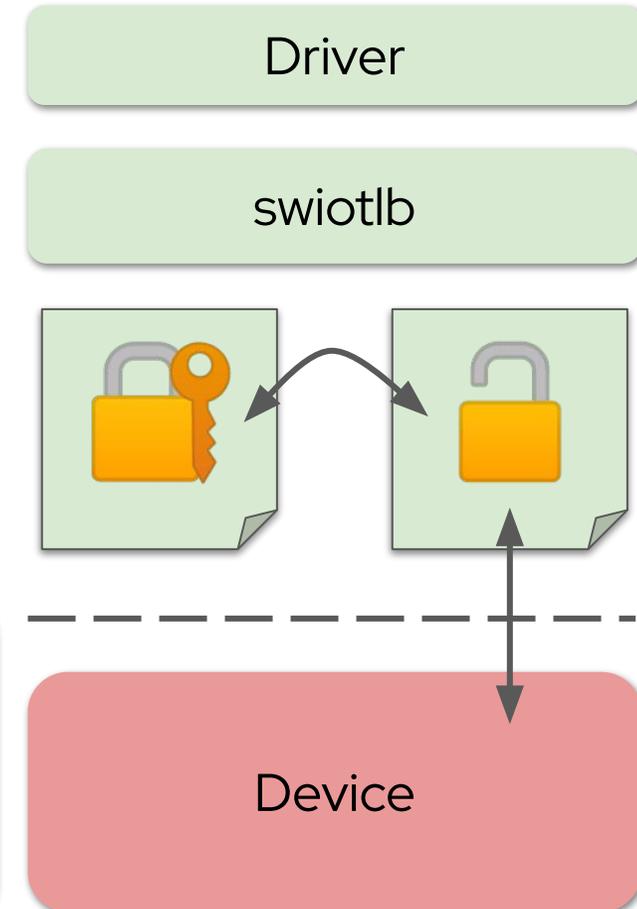
Place bounce buffer in unencrypted memory where untrusted devices can access it

Drivers already support this via Linux DMA

API - no driver effort needed!



Performance overhead & optimizations:
<https://www.usenix.org/conference/atc23/presentation/li-dingji>



Interrupt Hardening

Linux IRQ core by default enables IRQs immediately in `request_irq()`

What if a malicious device raises an IRQ when driver is not ready?

- ▶ Interrupt handler functions may execute in unprepared environment
- ▶ Need to protect against probe/remove race conditions in drivers

Previous patches didn't work with affinity managed interrupts, more work needed

Speculation Barriers

Drivers may be vulnerable to Spectre side-channel attacks

Hypervisor must not be able to extract data from Confidential VM

Even with encrypted memory, some attacks may be possible

Linux 6.6 *_nospec() macros not used in VIRTIO driver code

Area for future work

What about other Device Types?

- ▶ Serial - Cannot trust input untrusted device and cannot protect output from snooping
- ▶ GPU - Cannot trust output from snooping
- ▶ Input - Cannot trust input from untrusted device

What to do about these device types?

Trusted Devices

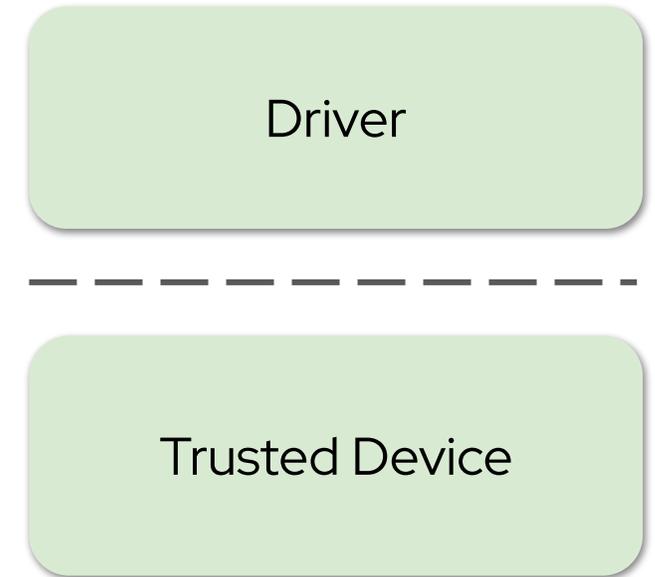
PCI-SIG TEE Device Interface Security Protocol (TDISP)

Attestation process verifies device is trusted

Trusted device becomes part of TCB

Bonus: swiotlb no longer needed (performance boost)

Goal: Only run on hardware/hypervisor configuration that guest considers trusted



Which Devices are Trusted?

User should be able to specify which implementations can be trusted

- ▶ “I trust Vendor A’s virtio-net device revision 2”
- ▶ “I don’t trust other virtio-net devices” or “Use swiolb for other virtio-net devices”

Not yet implemented, belongs in Linux driver core

User Policy on VIRTIO Devices & Features

Even minimal VIRTIO machines might need lots of code (PCI, ACPI)

Some users prefer virtio-mmio, others need PCI features (e.g. hotplug)

Some users may want to restrict some functionality (no hotplug)

Need a way to set this policy

User Policy for Confidential VMs

What to trust is a user policy decision

No single kernel parameter can express user's policy

Need control over:

- ▶ Which trusted devices are allowed
- ▶ Which untrusted devices are allowed
- ▶ What to do when malicious device action occurs?

Wish there was a comprehensive solution a la security modules or maybe Open Policy Agent support

Conclusion

Linux VIRTIO drivers have been hardened

Confidential computing story is work in progress:

- ▶ Untrusted devices are the first step
- ▶ Trusted devices in the future

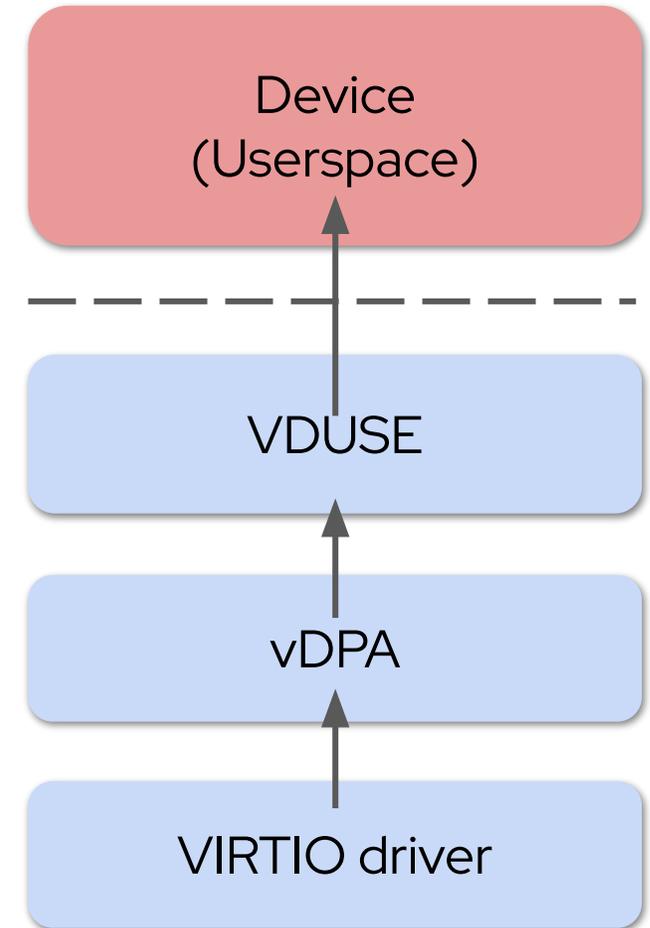
Need user policy controls for confidential VMs

Preventing Denial of Service in VDUSE

VDUSE userspace device must not hang system

VDUSE control plane uses read(2)/write(2) for asynchronous kernel->userspace communication

Existing drivers generally asynchronous and tolerant of unresponsive devices



What about using IOMMU inside guest?

Devices built into hypervisor don't benefit from vIOMMU isolation

vhost-user devices don't benefit from vIOMMU isolation

- ▶ vhost-user shared memory exposes all guest RAM



VIRTIO Intel TDX Status

Considered hardened according to Intel TDX document:

- ▶ virtio-pci with split virtqueue, no indirect descriptors
- ▶ virtio-block, virtio-net, virtio-console, virtio-9p, and virtio-vsock



<https://intel.github.io/ccv-linux-guest-hardening-docs/security-spec.html>