Virtualisation for the Masses:

Exposing KVM on Android

Will Deacon <will@kernel.org>

KVM Forum, October 2020

android

Introduction

Who am I and what am I hacking on?

- Active upstream kernel developer, co-maintaining the arm64 architecture port, locking, atomics, memory model, TLB, SMMU, ...
- Joined Android Systems Team at Google last year
- Leading the "Protected KVM" project to enable KVM on Android
  - Top contributors to KVM/arm64 for 5.9 and 5.10
  - Lots more to come (seems to be a hot topic)
- **Disclaimer: very much a work-in-progress! Upstreaming as we go.**

android

The state of modern Android

android

Generic Kernel Image (GKI)

- Problem: Separate kernel for each device does not scale and leads to fragmentation:
  - Difficult/expensive to provide updates
  - In-field release upgrades can be impossible
  - Bad for upstream

- GKI aims to maintain subset of kernel ABI within a given Android release and kernel version (e.g. `android11-5.4`)
  - GKI branches forked from android-mainline
    - Close to upstream
    - Updated with regular LTS merges
  - Vendors/OEMs can provide modules

https://source.android.com/devices/architecture/kernel/generic-kernel-image

android

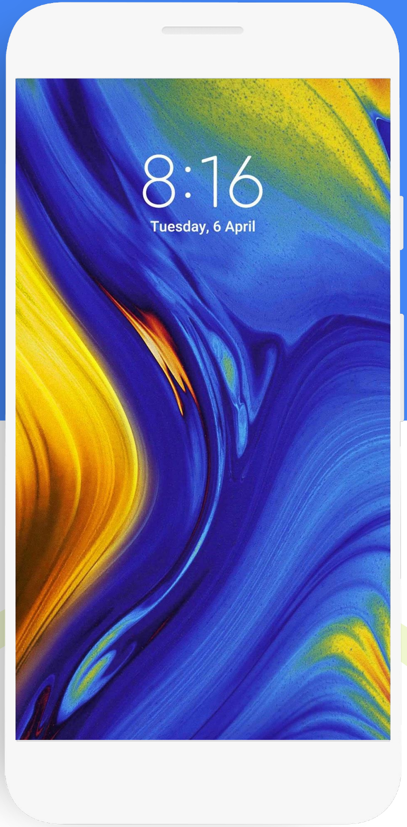
Virtualisation on Android today

tl;dr: It's the Wild West of fragmentation

When present, the hypervisor is treated as part of the device firmware and is typically supplied by the SoC vendor or OEM:

- Security enhancements for protecting the kernel
  - *"Mitigations are attack surface, too"*
    - Jann Horn, Project Zero
- Coarse-grained memory partitioning between devices using basic IOMMU-like hardware
- Running code outside of Android

Most of the time, there aren't even any virtual machines!

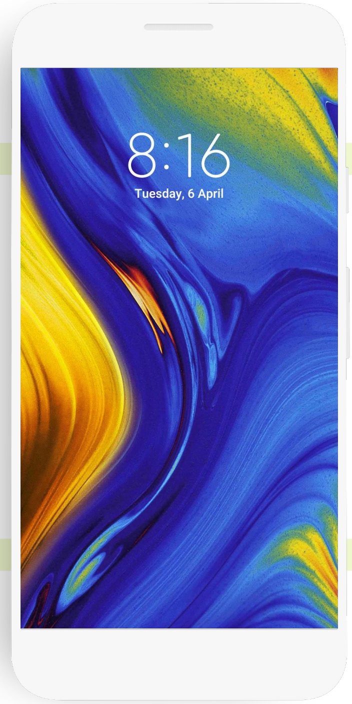


android

Security and functionality both lose out

Security

Increased TCB and difficulty/cost in providing streamlined updates across devices

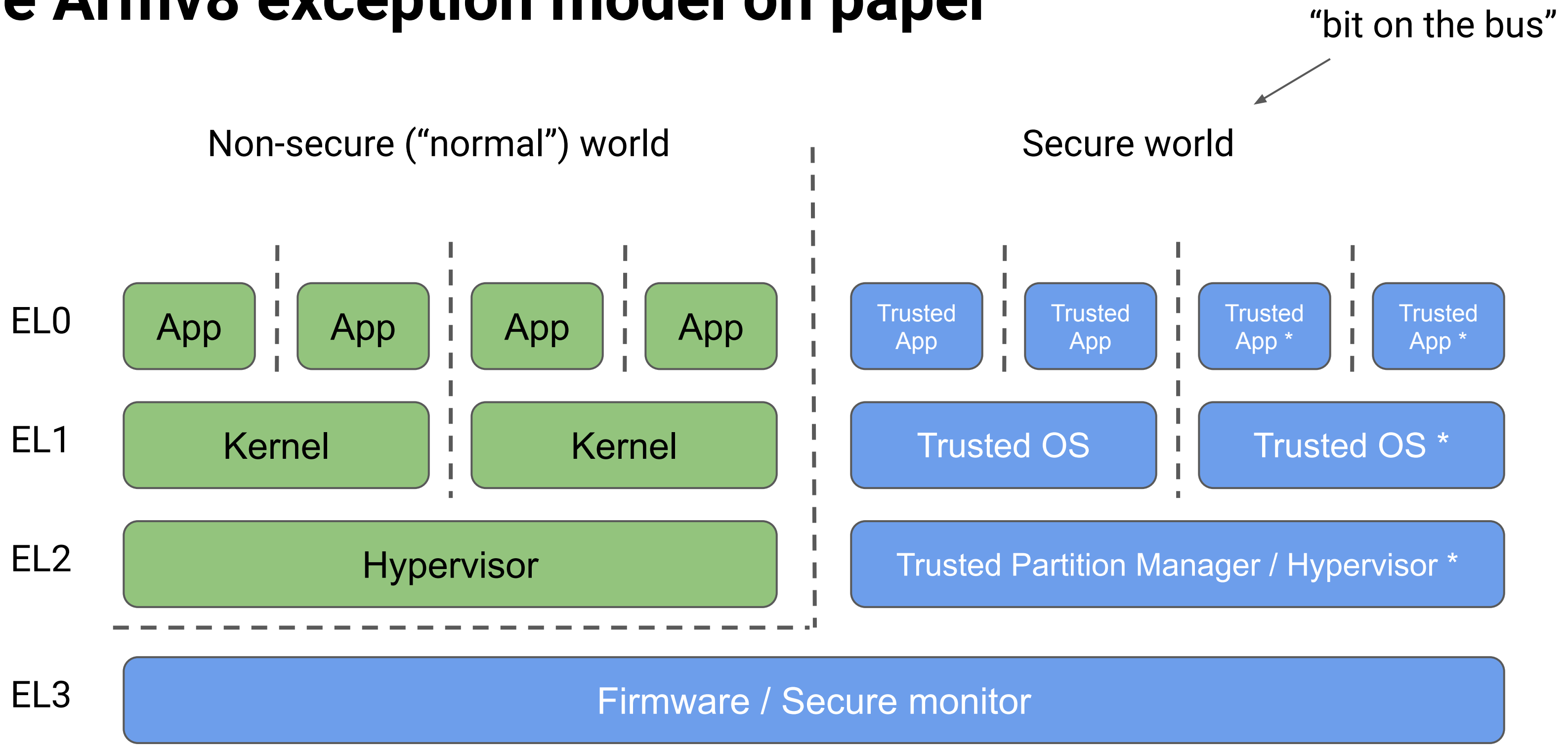


Functionality

Unable to leverage hardware virtualisation capabilities from within Android

android

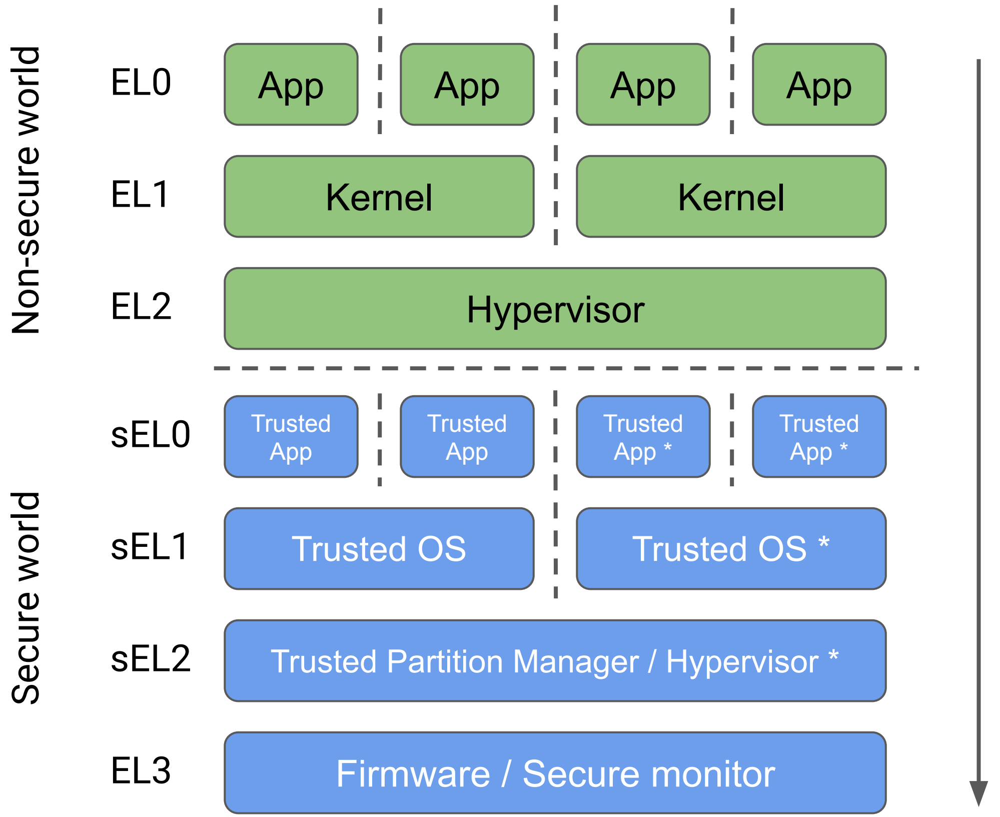
The Armv8 exception model on paper



* From Arm v8.4A

android

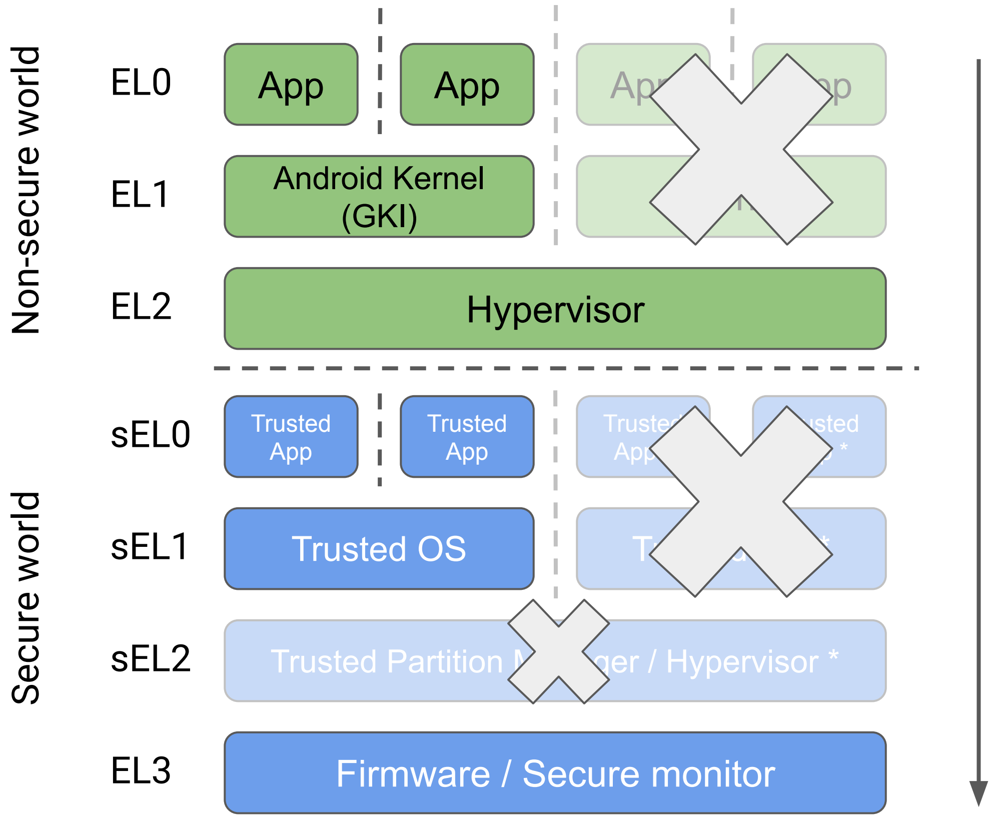
The Armv8 exception model sorted by privilege



Increasing privilege

android

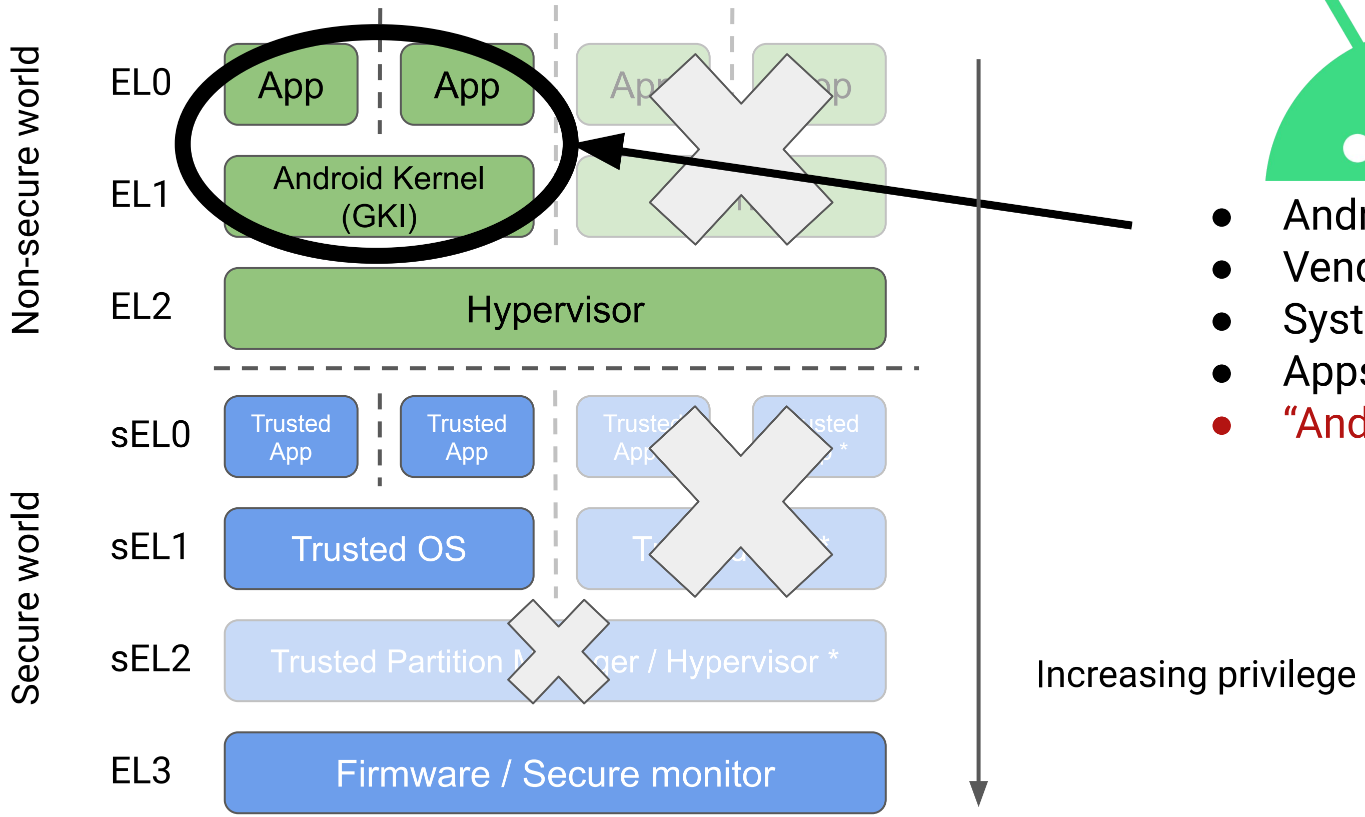
Mapping this to a modern Android system



Increasing privilege

android

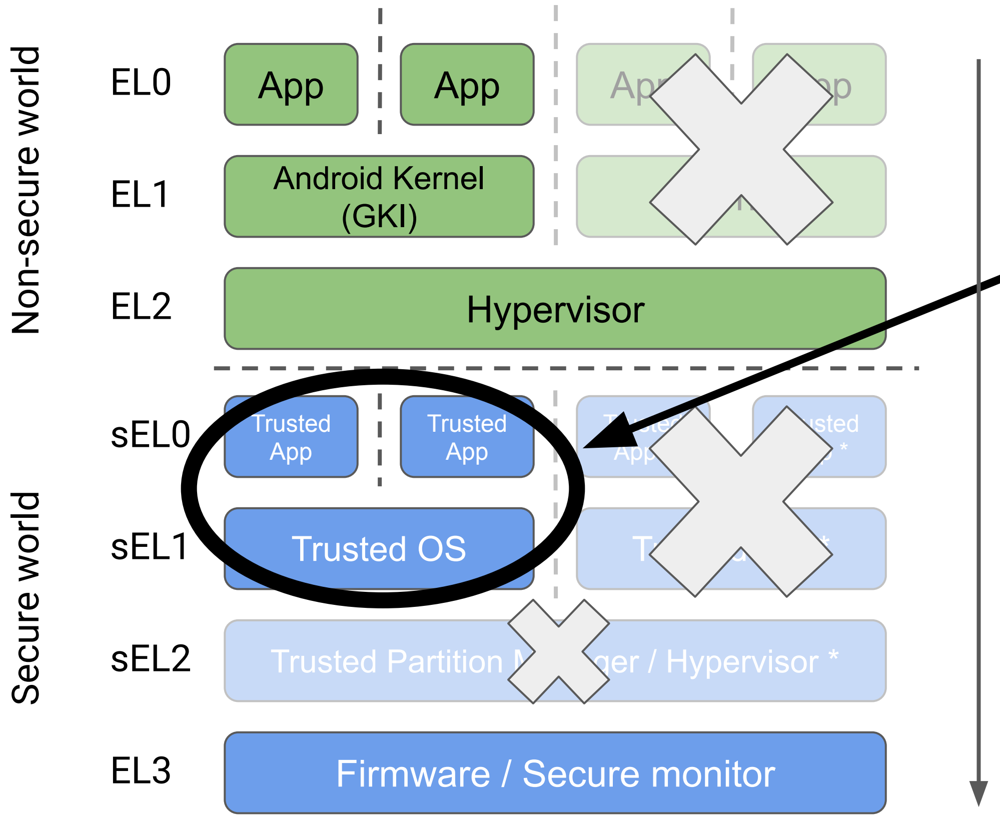
Mapping this to a modern Android system



- Android kernel (GKI)
- Vendor modules
- System and libraries
- Apps
- “Android”

android

Mapping this to a modern Android system



- DRM, crypto, ...
- Third party OSes
- Opaque blobs
- Per-device integration

Increasing privilege

android

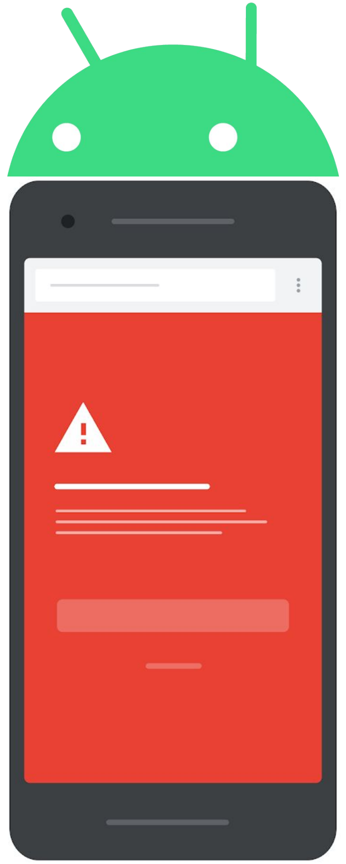
Trusted software

Definitions of trust (verb):

- *"To feel that something is safe and reliable"*
- *"To expect, hope or suppose"*

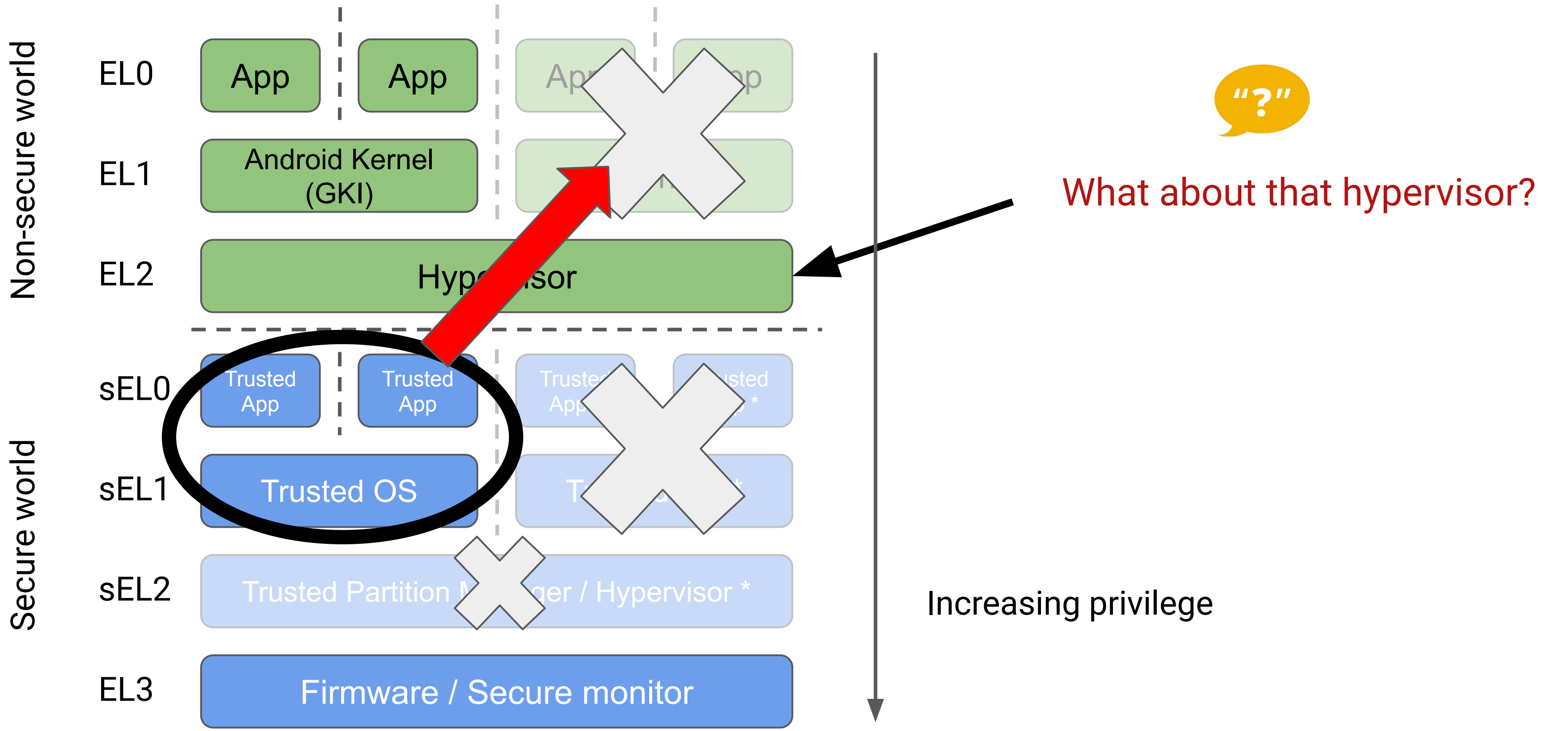
Android hopes that this software isn't malicious or compromised.

We need a way to de-privilege third party code and provide a portable environment in which to isolate services from each other and also from the rest of Android.



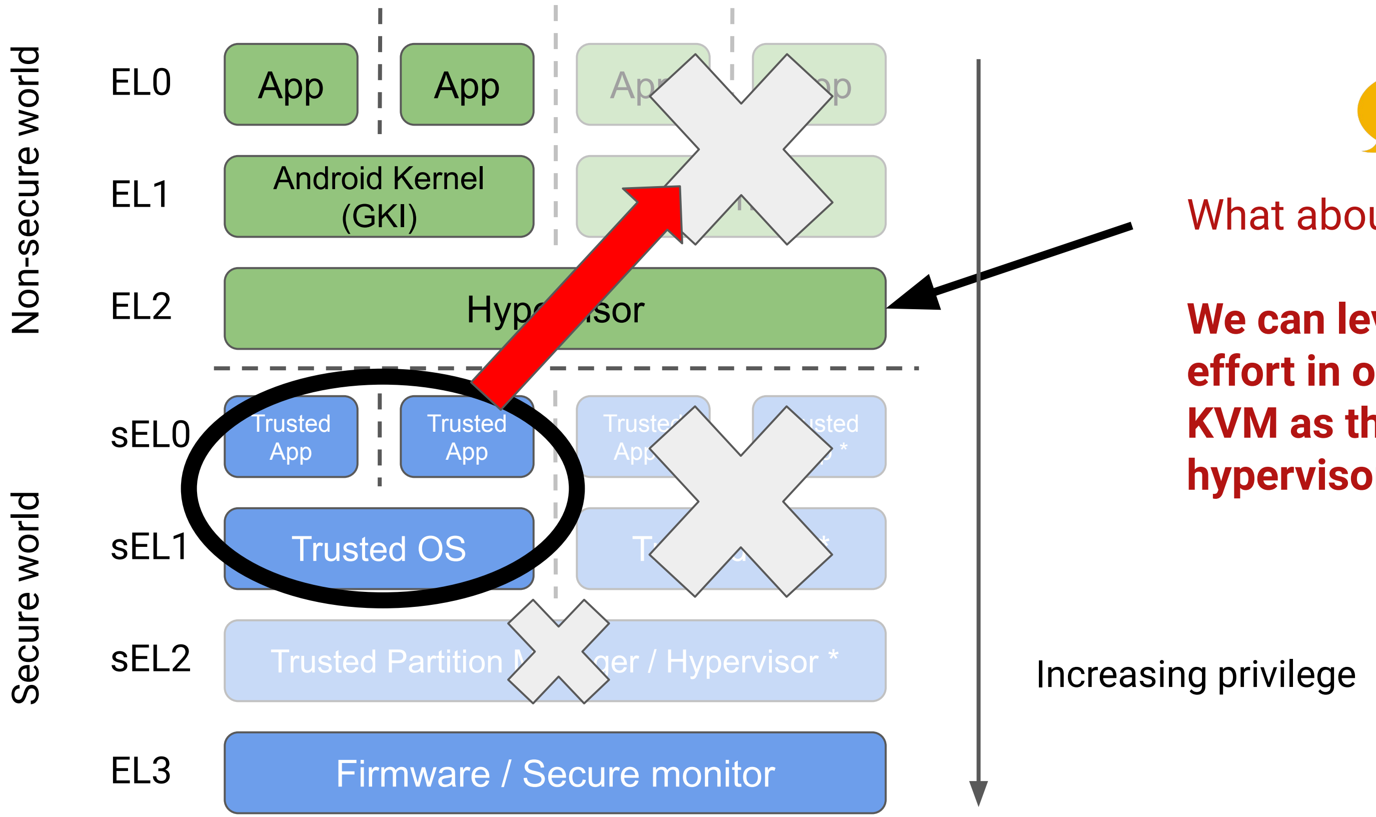
android

Virtualisation to the rescue?



android

Virtualisation to the rescue?



What about that hypervisor?

We can leverage the GKI effort in order to deploy KVM as the common hypervisor.

android

Virtualisation on arm64

(and how it’s used by KVM)

android

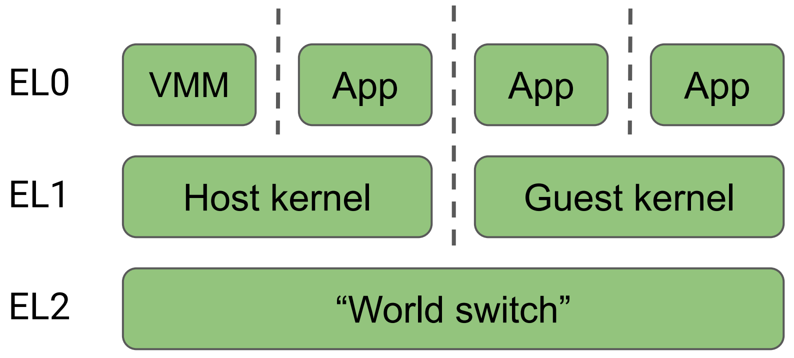
Virtualisation checklist

Availability	All arm64 Android devices have support for hardware virtualisation and a two-stage MMU.
Isolation	Stage-2 page-tables provide memory isolation
Security	Move third-party code out of EL2/Trustzone and into non-secure virtual machines
Portability	Implement a common hypervisor in Android enabling new applications and virtual machines that require confidentiality of data and integrity of computation.

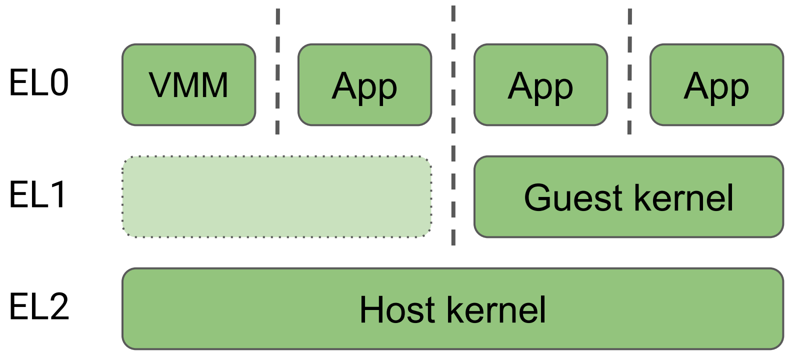
android

KVM on arm64

- Supported upstream on arm64 since v3.11
- Host kernel may reside at either EL1 or EL2:



nVHE (v8.0)



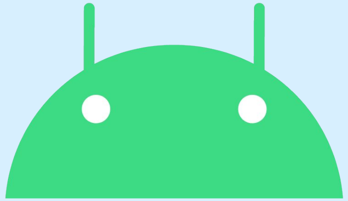
VHE (v8.1)
Blazingly fast!!1!

- Threat model places the entire host kernel (and VMM via `ioctl()`s) into the TCB; host has full access to guest memory.
    - This is a bit like “inverse Trustzone”

android

Big problem!

The threat model of Android is not aligned with the current design of KVM.



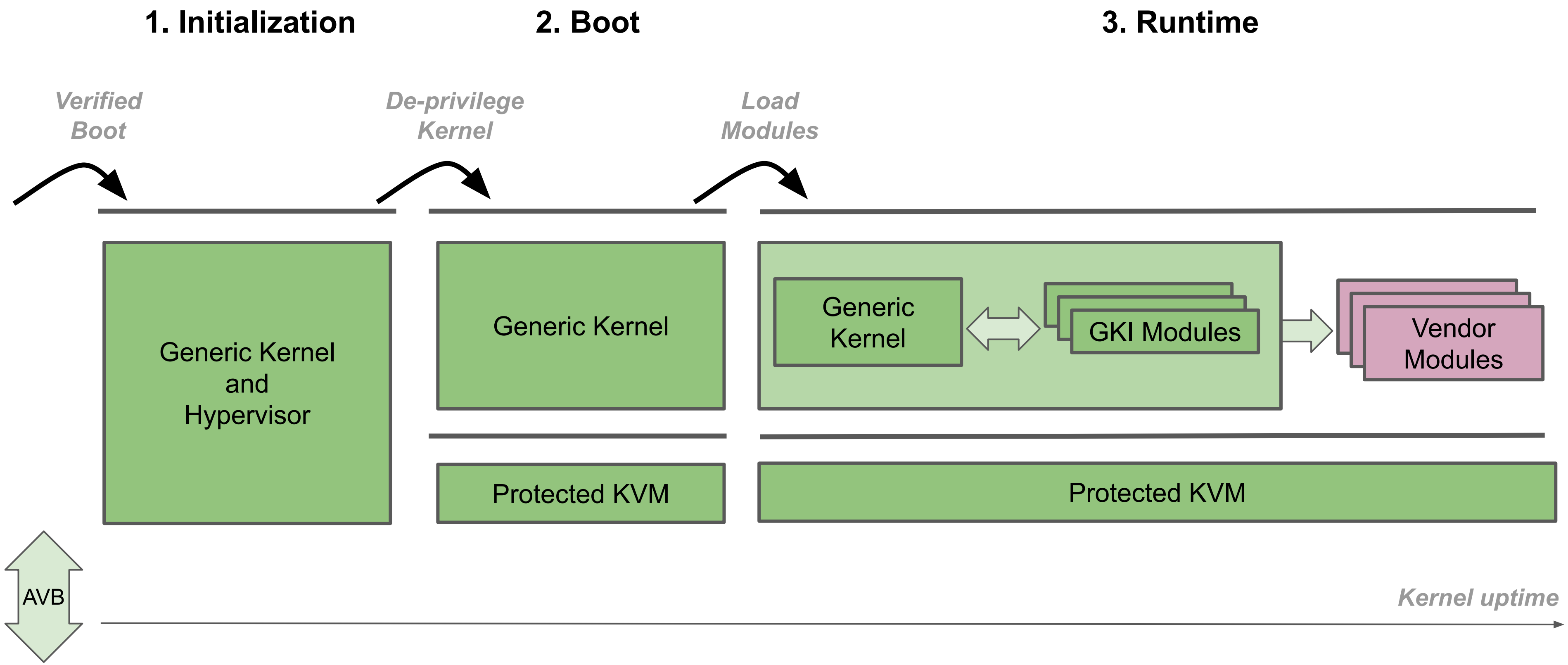
android

Revisiting nVHE with "Protected KVM"

Android's security model requires that guest data remains private even if the host kernel has been compromised. Maybe nVHE isn't so bad after all...

- Extend world-switch code at EL2 to manage stage-2 page-tables and guest state
- Install a stage-2 translation for the host kernel during boot before loading vendor modules
- Message passing between host and VM
- Template bootloader which accepts only signed VM images
- Formal verification techniques to reason about EL2 code

Q: Why not run Android in a VM instead?



Legend: 🟪 Vendor Specific 🟩 AOSP *AVB* - Android Verified Boot

android

The taming of EL2

Executing at EL2

The KVM nVHE EL2 environment is a pretty horrible place: it has its own limited virtual address space and cannot run general kernel code:

- Not preemptible/interruptible and unable to block/schedule
- Can access all of normal memory if mapped
- Very limited device access; typically no console
- Basically just context-switches EL1 and allows host kernel to run functions with elevated privilege
- Tight coupling with host kernel is optimal for KVM's threat model

Prior to 5.9, Linux offered `#define kvm_call_hyp(f, ...)` to run kernel functions annotated with `__hyp_text` at EL2.

android

Executing at EL2 (< 5.9)

```
// C code to run at EL2 (arch/arm64/kvm/hyp/tlb.c)
void __hyp_text __kvm_flush_vm_context(void)
{
    dsb(ishst);
    __tlbi(alle1is);
    if (icache_is_vpipt())
        asm volatile("ic ialluis");
    dsb(ish);
}

// EL2 entry dispatcher
kern_hyp_va     x0
do_el2_call                 // Indirect call to arbitrary address!!!
```

```
// EL1 hypercall (HVC #0)
#define kvm_call_hyp(f, ...)  __kvm_call_hyp(kvm_ksym_ref(f), ##__VA_ARGS__);

// Callsite
kvm_call_hyp(__kvm_flush_vm_context);
```

The EL2 object in 5.9/5.10

New threat model needs EL2 code to be self-contained & safe against compromised host kernel:

- Embed EL2 payload using separate ELF sections and symbol prefixing (similar to EFI stub)
- Fixed set of hypercalls rather than arbitrary function pointers
- Prior to de-privilege, host sets static keys and applies alternatives (one way switch)
- Following de-privilege, EL2 object no longer mapped for EL1

"Who needs namespaces when you have underscores?"

```
$ aarch64-linux-gnu-objdump -t -j .hyp.text arch/arm64/kvm/hyp/nvhe/kvm_nvhe.o
SYMBOL TABLE:
0000000000002000 g     F .hyp.text 00000000000000b4 __kvm_nvhe___host_exit
00000000000019e0 g     F .hyp.text 00000000000000c0 __kvm_nvhe___kvm_tlb_flush_vmid_ipa
0000000000004288 g     F .hyp.text 0000000000000048 __kvm_nvhe___vgic_v3_deactivate_traps
0000000000000048 g     F .hyp.text 00000000000000f0 __kvm_nvhe___sysreg_save_state_nvhe
00000000000043d0 g     F .hyp.text 0000000000000044 __kvm_nvhe___vgic_v3_init_lrs
0000000000001b30 g     F .hyp.text 0000000000000034 __kvm_nvhe___kvm_flush_vm_context
```

Symbol aliases created from "allowlist" of kernel symbols for use at EL2.

android

Executing at EL2 (5.9/5.10)

```
// C code to run at EL2 (arch/arm64/kvm/hyp/nvhe/tlb.c)
void __kvm_flush_vm_context(void)
{
    [...]
}

// EL2 entry dispatcher (now in C!)
switch (func_id) {
case KVM_HOST_SMCCC_FUNC(__kvm_flush_vm_context):
    __kvm_flush_vm_context();
    break;
```

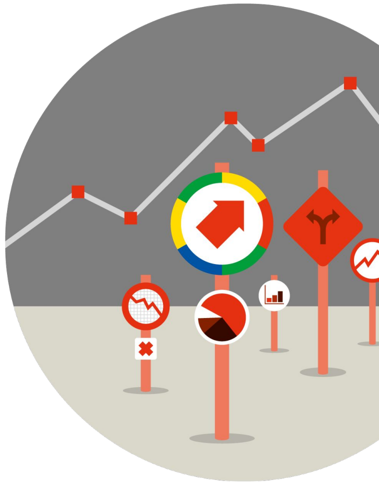
```
// EL1 hypercall (HVC #0)
#define __KVM_HOST_SMCCC_FUNC___kvm_flush_vm_context            2
arm_smccc_1_1_hvc(KVM_HOST_SMCCC_FUNC(f), ##__VA_ARGS__, &res);
#define kvm_call_hyp(f, ...)  kvm_call_hyp_nvhe(f, ##__VA_ARGS__);

// Callsite
kvm_call_hyp(__kvm_flush_vm_context);
```

Virtual memory at EL2 (without pKVM)

Today, the host kernel is trusted and therefore in control of the hypervisor virtual memory:

- Hypervisor stage-1 mappings created by the host
    - Hypervisor pages also mapped by the host linear mapping
    - KVM data structures (e.g `struct kvm`) mapped directly to EL2
    - More of the kernel gets mapped in over time! (no `hyp_unmap()`)
- Homebrew per-cpu implementation
    - Local CPU only
    - Directly reuses host per-cpu region
- Guest stage-2 page-tables also managed by the host kernel
    - Blindly installed by EL2 during VM world switch
- All page-tables constrained by host page-table configuration



Makes it trivial for a compromised host kernel to bypass new hypervisor restrictions.

android

EL2 MM bootstrap (5.11?)

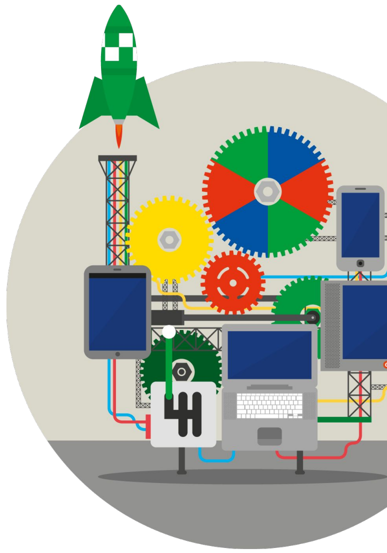
Allowing the host kernel to manipulate these page-tables breaks the revised security model:

- Page-tables must only be accessible to EL2
  - Stand-alone page-table walker merged for 5.10
  - http://lkml.kernel.org/r/20200911132529.19844-1-will@kernel.org

- Memory allocator needed for page-table pages
  - Hypervisor carveout donated from the host during boot
  - Trying to keep things as simple as possible
    - Host bootstraps EL2 prior to de-privilege
    - EL2 then transitions off temporary page-tables
  - https://android-kvm.googlesource.com/linux/+/refs/heads/pkvm

- Instantiate new per-cpu implementation at EL2 (also merged for 5.10)
  - https://lkml.kernel.org/r/20200922204910.7265-1-dbrazdil@google.com

- IOMMUs need to be kept in sync with the stage-2 page-tables



android

EL2 MM bootstrap (5.11?)

```
#define hyp_vmemmap ((struct hyp_page *)__hyp_vmemmap)
struct hyp_pool {
        nvhe_spinlock_t lock;
        struct list_head free_area[HYP_MAX_ORDER + 1];
};

// Buddy allocator for page allocation at EL2
void *hyp_alloc_pages(struct hyp_pool *pool, gfp_t mask,
                      unsigned int order);
```

```
// kvm_hyp_setup() [Complete bootstrap and de-privilege host]
kvm_call_hyp_nvhe(__kvm_hyp_setup, base, kern_hyp_va(__va(base)), size,
                  kvm_get_bp_vect_pa(), num_possible_cpus(),
                  kern_hyp_va(per_cpu_base));
free_hyp_pgds();

// cpu_init_hyp_mode() [Install host vectors with temporary MMU setup]
vector_ptr = kern_hyp_va(kvm_ksym_ref(__kvm_hyp_host_vector));
arm_smccc_1_1_hvc(KVM_HOST_SMCCC_FUNC(__kvm_hyp_init), pgd_ptr, tpidr_el2,
                  hyp_stack_ptr, vector_ptr, &res);
```

Stage-2 for the host

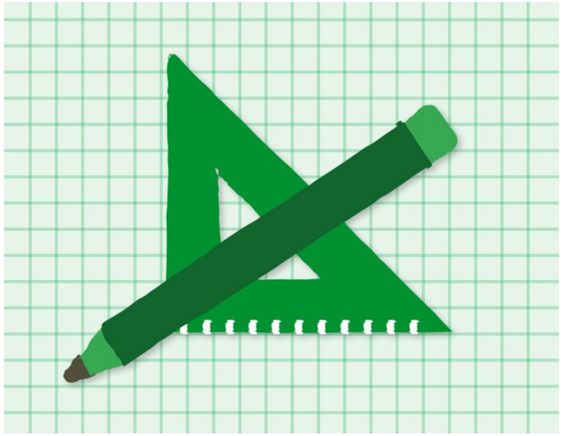
- Memory will disappear from the host as it is assigned to a guest
  - "KVM protected memory extension" from Kirill Shutemov
  - Inject stage-1 abort or treat as RAZ/WI?
- IOMMU support
  - Unfortunate reliance on SoC design and sensible hardware
- Kernel self protection?
  - Looks to the host like the permissions have changed for *physical* memory
  - Allow host to change permissions for RW memory it owns?
- VMM will not be able to access guest state (including CPU registers and memory)
  - Negotiate shared memory regions with guest for virtio
  - **Q: How is a guest initialised to begin with?**

android

Template bootloader

Requiring VM images to reside in pre-populated carve-outs doesn't scale…

- … but we also need to ensure that guest payloads haven't been tampered with by the host
- Small first-stage bootloader installed in carve-out memory during host boot
  - Exploring bare-metal rust implementation
- Accessible only to EL2 and used as initial entry point for protected guests
  - Performs signature check on guest payload
  - **Q: Does this really need to be arm64-specific?**



android

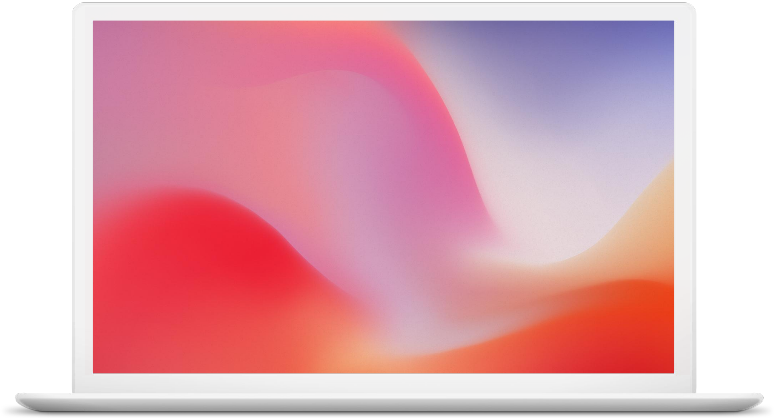
The virtual platform

android

Adapt crosvm as the VMM

Reuse the Chrome OS Virtual Machine Monitor:

- Part of ChromeOS and now included in AOSP
- Lots of related talks at KVM Forum!
- Modern codebase written in Rust
- Focus on security and sandboxing
- Many virtio devices implemented
- Cross-architecture (surprisingly important!)
  - https://source.android.com/setup/create/cuttlefish



We provide a fairly basic arm64 virtual platform:

- Fixed memory map
- CPUs onlined via PSCI calls
- Arm architected timer
- RNG/entropy service
- **PV interrupt controller (rVIC) [Marc Zyngier's talk]**
- What about I/O?

android

Just use virtio, stupid!

Virtio is the best thing since sliced bread and we should just use it for everything.
Job done?

- Strong desire to avoid changes to the spec
  - MMIO traps to the hypervisor
  - Re-use existing device/driver implementations

- Guest must use crypto (e.g. fs-verity) as host can intercept data due to lack of hardware memory encryption

- No shared-memory device?
  - Virtio assumes guest memory is shared with the host

https://www.kernel.org/doc/html/latest/filesystems/fsverity.html

android

Bounce-buffering via shared windows

The Virtio 1.1 specification introduces the `VIRTIO_F_ACCESS_PLATFORM`reserved feature bit:

“[...] indicates that the device can be used on a platform where device access to data in memory is limited and/or translated.”

When set, causes a Linux guest to use the DMA API for virtio allocations.

- We can force the use of bounce-buffers by passing `swiotlb=force`
- We then just need to allow the host to access the bounce buffer pages
  - Expose SHARE/UNSHARE hypercalls to the guest to update host stage-2.
  - Hook the `set_memory_{decrypted,encrypted}()` API to share/unshare bounce buffer pages

android

Zero-copy transfers using shared memory

Bounce buffers force the copying of all I/O data through a shared window:

- This is fine for many use-cases, but introduces undesirable overhead/incompatibility for others
  - e.g. Binder shared memory
- Need a handshake to share memory from host to guest
  - Don't want the guest to have access to all of host memory
  - Don't want the host to silently change guest stage-2
- Arm FF-A specification aims to solve this problem but is fairly heavyweight and not cross-architecture

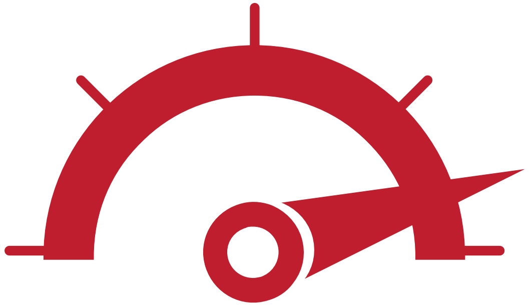
Q: What should we be using here?

android

What’s next?

Still loads to do:

- Complete mm bootstrap
- Stabilise user ABIs
- Settle on solution for zero-copy I/O
- Move more guest state up to EL2
- Memory poisoning
- SMC proxying
- Attestation
- Ballooning
- Integration with rest of Android
- Continue upstreaming...



android

Questions?

<android-kvm@google.com>