Google

Pocket-sized virtual machines

Booting small Linux payloads in Android

Android Security

Pierre-Clément Tosi (ptosi@google.com)

Embedded Recipes - Sep 2023

Agenda

01 Android Virtualization Framework

02 Microdroid: a small guest kernel

03 Protected Virtual Machine Firmware

04 Next Steps

05 Q&A

01

Android Virtualization Framework

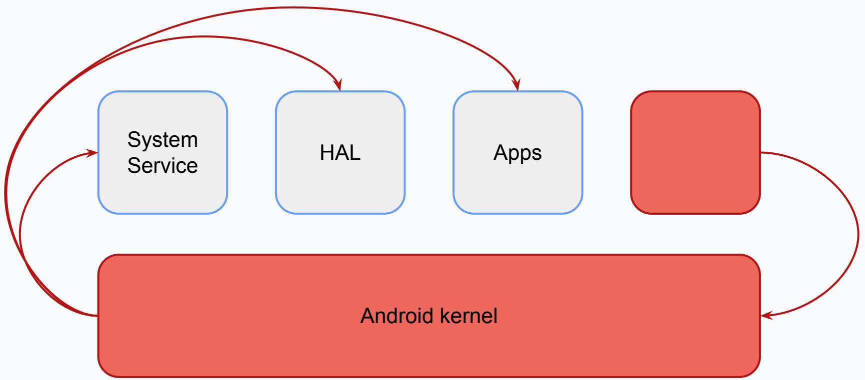
Android: Privilege Escalation

Android userspace runs

- System services
- Hardware abstraction layers
- System Applications
- Third-party applications
- …

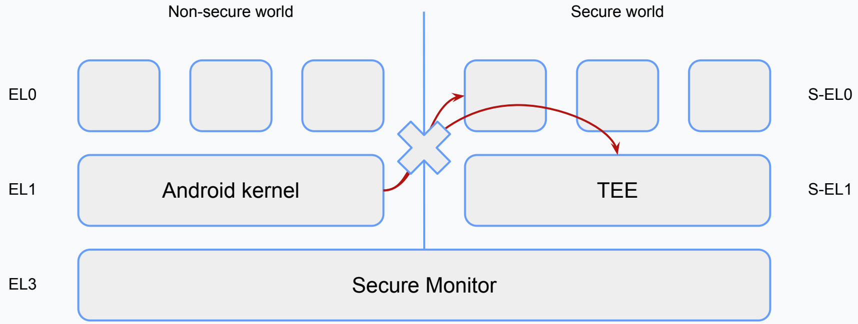
And their interface with the kernel is *large* and *complex*.

A malicious process may exploit a vulnerability in the kernel …
… to achieve privilege escalation and attack other processes …
… compromising user data and privacy!



Arm TrustZone: Isolation

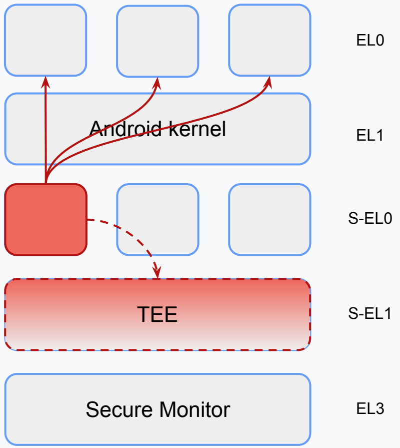
The Arm architecture defines isolated CPU modes, *worlds*:



Even if the Android kernel gets compromised, it is still prevented from accessing anything running in *secure*!

So should we move all our privacy sensitive applications there?

TZ was designed for security-critical firmware, so in practice:



If an evermore complex *trusted application* gets compromised, it is (probably) able to attack anything running in *non-secure*!

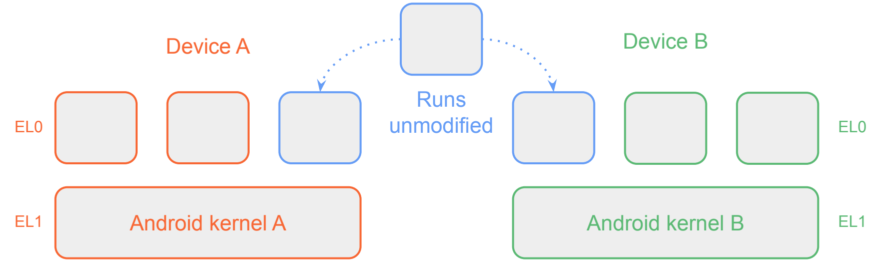
No.

Arm TrustZone: Fragmentation

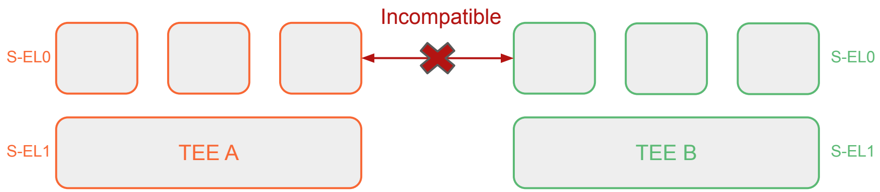
Unlike Android kernel patches[1], TZ updates are often a **time-consuming** and **costly** process, creating a tradeoff between user security and device maintenance cost.

TEEs are naturally less **feature-rich** than modern OSes such as Android so that tools and libraries available are more limited than for Linux.

An Android application is compatible with all Android devices



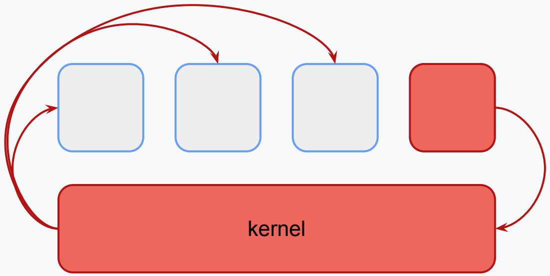
The TrustZone ecosystem is much more **fragmented**



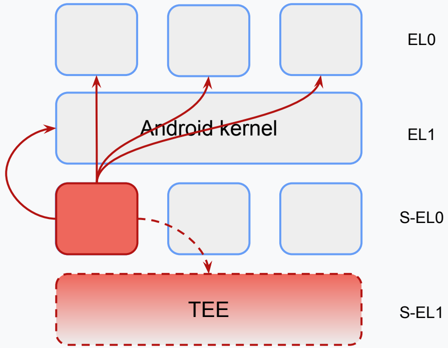
[1] see Android Generic Kernel image

Android Virtualization Framework: Motivation

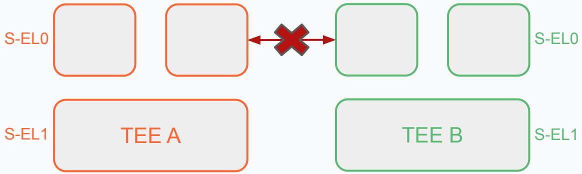
Privilege Escalation



Privilege vs Isolation



Fragmentation

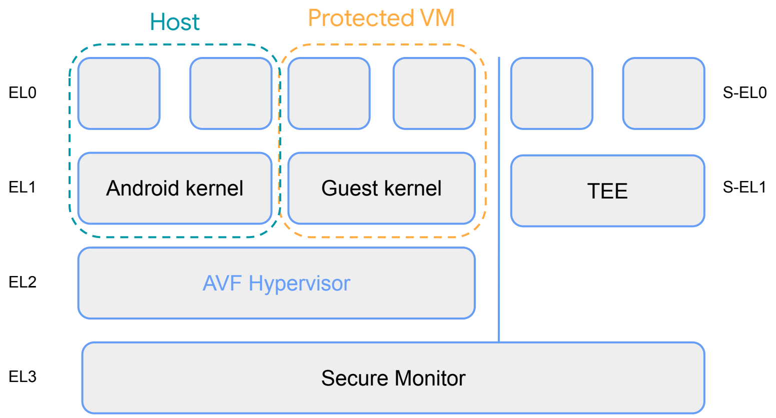


Less feature-rich than Linux

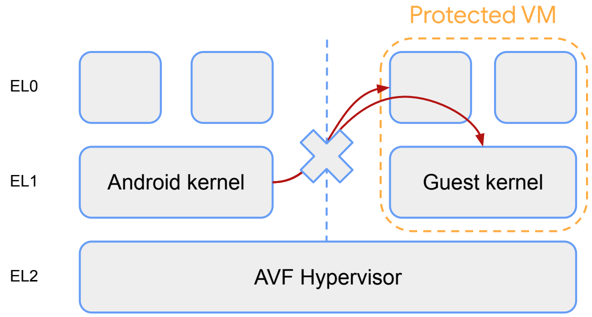
Harder to update than Android

Android Virtualization Framework

AVF introduces a hypervisor isolating Android from *protected VMs*



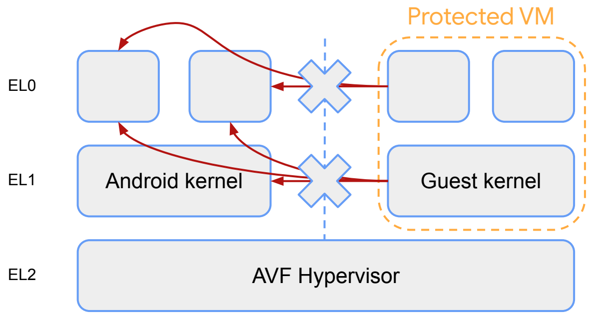
No attacks from compromised kernels on protected payloads



Android Virtualization Framework

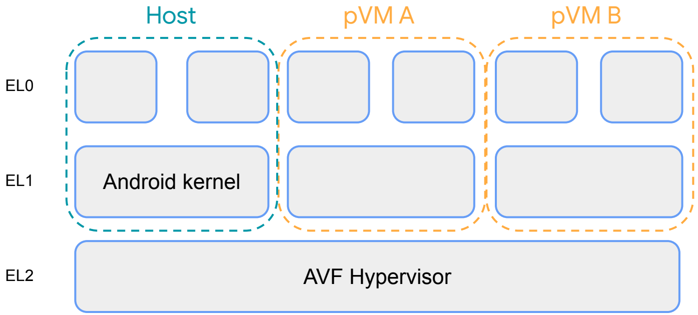
Deprivileged

Apps in pVMs are unable to compromise Android



Flexible

The hypervisor can create multiple pVMs



Standardized

- pVMs can run feature-rich environments, similar to Android
- pVM payloads may be distributed in a device-agnostic way

Secure

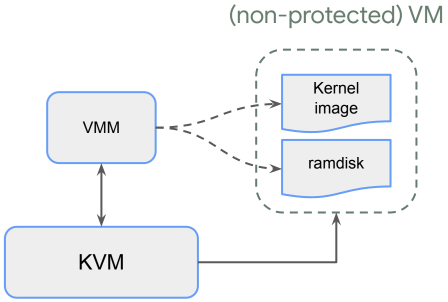
- Hypervisor is purpose-built, small and straightforward
- Hypervisor can be part of the Android Generic Kernel image
- **Protected KVM** is open-source (AOSP) and upstreamed[1] to Linux

[1] Some has been merged, the rest is under review or on its way

Google

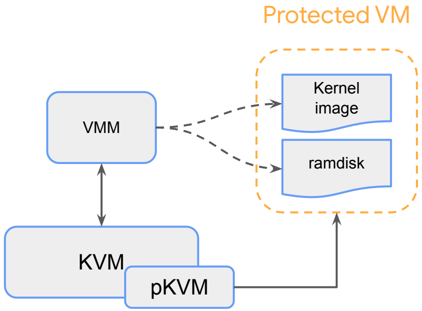
AVF Virtualization Model

AVF follows the KVM architecture:



- Guests are spawned **dynamically** by the host
- The virtual platform is implemented in **userspace** (Virtual Machine Monitor), which preloads the guest
- Host **schedules** guests as regular processes

pKVM extends the existing KVM infrastructure:



- Hypervisor prevents the host from accessing guest memory after pVM boot: the host ***donates*** it
- **Denial-of-service attacks** from the host are accepted!

02

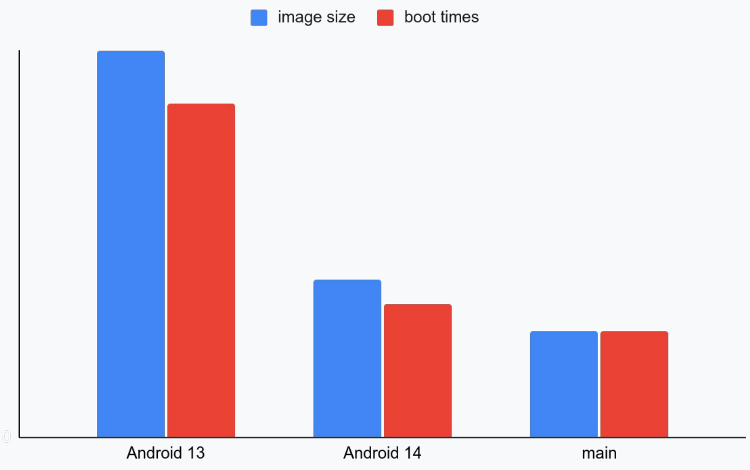
Microdroid

Microdroid

AVF can run any guest kernel: we maintain **Microdroid**.

- Built for pVMs (only)
- Stripped down version of Android
  - NDK APIs (Bionic syscalls)
  - APKs & APEXes
  - Binder RPC (vsock)
  - Verified Boot, SELinux, dm-verity
  - ADB, logcat, tombstone, GDB
  - No Java[1], Zygote, graphics, or HALs

Loads and executes an APK payload (binary + shared libraries)



(defconfig, prebuilts)

[1] Intentional but we have workarounds, if necessary

Google

03

Protected VM Firmware

Protected VM Firmware

“Host is not trusted” but “VMM configures pVM”?

We need a trusted entity to

- Verify that the loaded **guest images** were not tampered with
- Validate that the **virtual platform** was properly configured

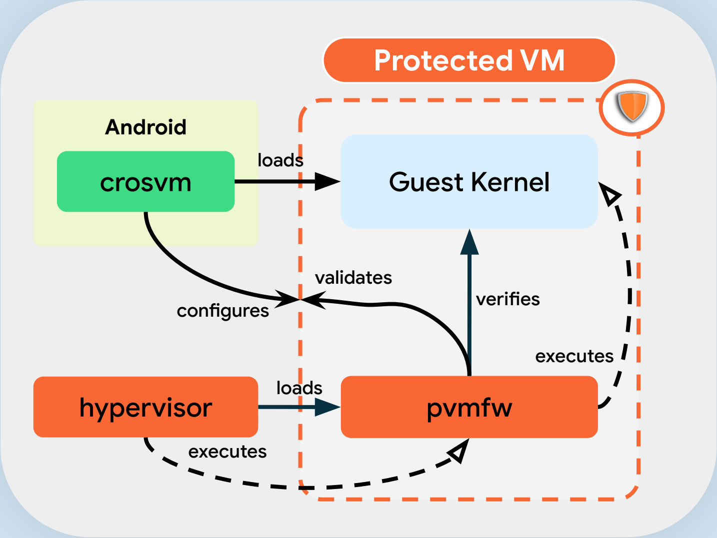
Doing that in the hypervisor would increase its attack surface.

Instead, a **trusted** piece of software implementing these, the pVM firmware (“pvmfw”), is loaded from **reserved memory** into guest memory and acts as the **pVM entry point**.

If verification fails, it **aborts the boot process**.

Trust model of pvmfw:

- As trusted as the hypervisor by the guest kernel
- As trusted as the guest kernel by the hypervisor
- As trusted as the guest kernel by the VMM

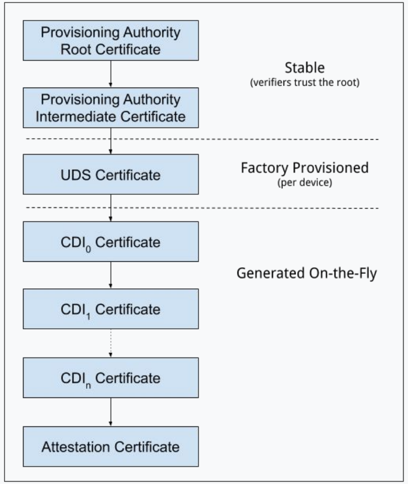


Google

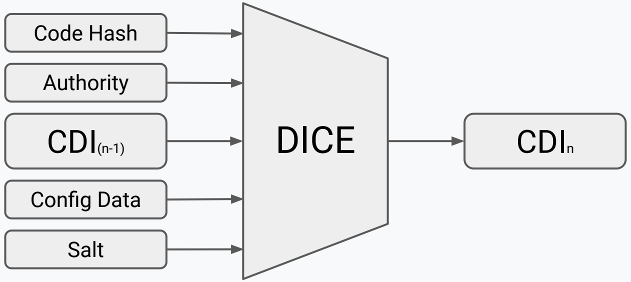
DICE for protected VMs

DICE chains attest of a boot sequence and provide each stage with a **certificate** and a **private key** (CDI), see

- Trusted Computing Group (TCG) - Hardware Requirements for a Device Identifier Composition Engine (DICE)
- Google - Open Profile for DICE
- Android - Android Profile for DICE



Each stage adds a layer to the chain through cryptographic operations:



The reserved memory loaded into the pVM by the hypervisor contains a **DICE chain**, which pvmfw extends with the measurements from its **verified boot**, producing a new chain that the guest can make use of to:

- Perform cryptographic operations with a key that was protected from the host
- Attest of its identity to external entities
- Derive new chains for its payloads (*e.g.* user-space applications)

Before executing the guest kernel, pvmfw wipes the memory where its private key was stored.

pvmfw: Reference Implementations

The virtual environment of pvmfw makes it *very* similar to a baremetal bootloader as it must

- Manage its own **runtime environment**: set up its stack, BSS, heap, relocate .data, …
- Manage the **system** (system registers, page tables, exceptions, caches, …)
- Implement its own **device drivers** (virtio-pci, virtio-blk, UART)
- Track the **pre-populated contents** of RAM (*e.g.* guest kernel, initrd, DT)
- Run from a **restricted region** (4MiB) of virtual memory

… but must also follow the **AVF threat model** *i.e.* must distrust the virtual platform!

In Android 13, pvmfw was a custom target of AOSP U-Boot (2022.01) (source & prebuilt)

From Android 14, pvmfw was re-written from scratch (source)

Google

Protected VM Firmware in Android 14

In Android 14, the pvmfw project is

- Part of the AOSP codebase (Apache License 2.0)
- Fully integrated with the Android Build system (no external build required)

Written in **Rust**[1]

- Contributions beyond Android
  - The aarch64-paging and smccc crates were published to crates.io
  - The virtio-drivers crate was extended to support PCI and match the AVF model
- Only using C/C++ in industry-standard libraries (libfdt, BoringSSL, libopen-dice)

[1] see *Memory Safe Languages in Android 13*

Next Steps

AVF

Provide more functionality to support new use-cases (device passthrough, VM-to-TZ channels, ...)

Hypervisor (pKVM)

Support new use-cases (protected IOMMU drivers), Improve performance (CoW, optimised ABI)
&
Continue the upstreaming effort

pvmfw

Implement any security-sensitive functionality required by new use-cases (anti-rollback protection, device validation, ...)
&
Further optimise boot times as needed

You!

Read more about AVF:
source.android.com

Give it a try:
Apps/custom VMs in AVF

Get in touch:
android-kvm@google.com