Agenda

• Security and functional goals of the Arm CCA software architecture
• Key components and their roles in the architecture
• Concepts and flows
• Impacts on host and Realm software
• Standardization
Goals

- Protect confidentiality and integrity of Realms
  - From other Realms
  - From all Non-secure software (including the hypervisor)
  - From all Secure software
- Enable user to establish trustworthiness of a Realm and the underlying platform
- Retain ability of system software (hypervisor) to manage resources used by Realms
- Avoid imposition of arbitrary resource limits
  - Number of live Realms should be limited only by available memory
- Minimize size and complexity of software inside a Realm’s TCB
  - Simplify reasoning about security properties
Goals

- Protect confidentiality and integrity of Realms
  - From other Realms
  - From all Non-secure software (including the hypervisor)
  - From all Secure software

- Enable user to establish trustworthiness of a Realm and the underlying platform

- Retain ability of system software (hypervisor) to manage resources used by Realms

- Avoid imposition of arbitrary resource limits
  - Number of live Realms should be limited only by available memory

- Minimize size and complexity of software inside a Realm’s TCB
  - Simplify reasoning about security properties

We’ll come back to these ...
Components and roles

Goal: Minimize size and complexity of software inside a Realm’s TCB

![Diagram showing components and roles]

- Non-secure state:
  - EL0: VM
    - App
  - EL1: Kernel
  - EL2: Hypervisor
  - EL3: Monitor

- Secure state:
  - EL0: TOS
  - EL1: SPM

Hardware
Components and roles

Goal: Minimize size and complexity of software inside a Realm’s TCB
Components and roles

Goal: Minimize size and complexity of software inside a Realm’s TCB
Components and roles

Goal: Minimize size and complexity of software inside a Realm’s TCB
Components and roles

Goal: Minimize size and complexity of software inside a Realm’s TCB

Realm Management Monitor (RMM)
- Provides services to Host and Realm
  - Contains no policy
  - Performs no dynamic memory allocation
- Realm Management Interface (RMI)
  - Create / destroy Realms
  - Manage Realm memory, manipulating stage 2 translation tables
  - Context switch between Realm VCPUs
- Realm Services Interface (RSI)
  - Measurement and attestation
  - Handshakes involved in some memory management flows
- Trusted PSCI implementation

Monitor
- Context switching CPU execution between security states
- Management of Granule Protection Table (GPT)
  - Controls assignment of memory to a Physical Address Space (PAS)
Memory delegation

Goal: Avoid imposition of arbitrary resource limits

- Each Physical Address Space (PAS) is an aliased view onto the underlying memory
- The system architecture defines a Point of Physical Alias (PoPA), which is where these views are reconciled
Memory delegation

- At system start-up, memory is reserved by boot firmware for use by the RMM

Realm PAS

Non-secure PAS

DRAM

Initial carve-out for RMM code and static data

Carve-out described to Host via firmware tables
Memory delegation

- An attempted access to Realm PAS memory results in a **Granule Protection Fault**
  - This is a new type of architecturally-defined exception
  - The fault is delivered to the accessor, or can be trapped to a higher Exception Level
Memory delegation

At runtime, the Host can request memory to be delegated to the Realm PAS.
Delegation is performed at page granularity.
The Host is responsible for tracking these “holes” in its own PAS, and avoiding access.

Goal: Retain ability of system software (hypervisor) to manage resources used by Realms.
Memory delegation

- The Host can request construction of **RMM objects** from delegated memory
  - Realm Descriptor (RD) – stores attributes of a Realm
  - Realm Translation Table (RTT) – stage 2 translation table for a Realm
  - Realm Execution Context (REC) – stores state of a Realm VCPU
  - Data – memory (code / data) which can be mapped into a Realm’s address space
Memory delegation

- Realm PAS memory which is not in use by the RMM or by a Realm can be reclaimed by the Host
- During this *undelegation* operation, memory contents are scrubbed
Memory delegation

- Physically-contiguous delegated memory can be mapped into a Realm as a block
- Existing hypervisor mechanisms for optimizing TLB usage can be applied to Realms
Realm lifecycle

Create
- Allocate metadata
- Provide configuration parameters

Populate
- Create RECs (VCPUs)
- Create RTTs (stage 2 TTs)
- Add initial (measured) Realm content

Activate
- No further changes to Realm configuration or contents permitted by Host

Run

Destroy
Realm lifecycle

Create
- Allocate metadata
- Provide configuration parameters

Populate
- Create RECs (VCPUs)
- Create RTTs (stage 2 TTs)
- Add initial (measured) Realm content

Activate
- No further changes to Realm configuration or contents permitted by Host

Run

Destroy

Goal: Enable user to establish trustworthiness of a Realm and the underlying platform

- During execution, Realm can request an **attestation report** from the RMM
  - This includes a description of the state of the platform, and of the initial state of the Realm
  - Based on this report, a user can decide whether to place trust in the Realm
  - More on this from **Simon Frost**, in the next session ...
Realm context switching

- VCPU context is saved in an RMM-managed data structure (REC)
- VCPU scheduling policy remains in the hypervisor
  - Realm entry occurs in response to a request from the Host
- On Realm entry, RMM restores VCPU state from the REC
- On Realm exit
  - RMM saves VCPU state to the REC
  - Sanitized syndrome information is returned to the hypervisor via a shared data structure
- Overall scheduling flows and ABIs are unchanged
  - Run loop
  - ESR-driven VM exit flow

Goal: Retain ability of system software (hypervisor) to manage resources used by Realms
Realm memory

- IPA space partitioned at Realm creation into **protected** and **non-protected** ranges

- **Protected locations**
  - Initial contents are measured
  - Initially-unpopulated locations can be lazily populated with uninitialized memory
  - Realm instructions can only be fetched from protected memory

- **Memory backing the PAR must be pinned by the Host**
  - Protected paging of Realm memory will be added in future

- **Unprotected locations**
  - Can map to NS memory
    - Expected to be used with PV I/O, e.g. virtio
  - Accesses can be emulated by Host
Realm interrupts

- Arm CCA doesn’t guarantee Realm availability
- Delivery of virtual interrupts to a Realm is supported
- Hardware and software change is minimized
  - RME makes no changes to GIC hardware architecture
  - Realm interrupt control is via a Host-emulated vGIC
  - Interrupts are presented to a Realm via GIC CPU interface
- No security guarantees are provided regarding Realm interrupts
  - A malicious Host is able to
    - Suppress interrupt delivery
    - Deliver spurious interrupts
    - Deliver interrupts to the wrong VCPU
- In-Realm GIC driver must be hardened against a malicious GIC emulation

Emulated by non-trusted Host
- Per-interrupt lifecycle state (Pending / Active)
- Per-interrupt configuration
  - Priority
  - Routing

Protected by RMM
- CPU-level mask controls
  - PSTATE.{I,F}
  - IC{C,V}_PMR_EL1

© 2021 Arm
Summary of Host software impacts

• Generally confined to hypervisor / VMM
  • No impact to Host firmware (UEFI / ACPI / DT)

• VM creation flow
  • VMs and VCPUs represented by RMM objects

• Memory management
  • RMM-managed stage 2 translation tables
  • Swapping / merging of Realm protected memory is not supported

• Management of hardware state
  • RMM-managed VGIC, timers, PMU, debug, VFP / SVE
  • RMM-managed Realm entry / exit
Summary of Realm software impacts

• Firmware interfaces for measurement handover
• Sharing Non-secure memory with Host
• Hardening against malicious hypervisor
  • Malicious emulation of devices (e.g. GIC, virtio)
    – Interface level
    – State-machine level
• Provocation of races and timeouts (e.g. SMP bring-up)
• Long tail of potential threats to consider in future
Standardization

- RMM architecture specification defines
  - Realm Management Interface
  - Realm Services Interface
  - Realm exception model
  - Realm execution environment
    - Memory
    - Interrupts
    - Timers
  - Realm measurement contributions
- Interface between Monitor and RMM is IMPDEF
Summary

- Arm CCA introduces a new firmware component, the Realm Management Monitor
  - The RMM exposes interfaces to the Host and to Realms
- Resources required by the RMM and by Realms are managed by the Host
- Realm creation includes a temporal boundary beyond which security guarantees apply
- Realm IPA space is partitioned into protected and non-protected regions
- Policy for context switching remains in the Host; mechanism is delegated to the RMM
- A Realm must protect itself against a malicious interrupt controller emulation
- The RMM implementation is tightly coupled with the Monitor implementation
Thank You
Danke
Gracias
谢谢
ありがとうございました
감사합니다
धन्यवाद
شكرًا
ধন্যবাদ
תודה