arm

Compartments and other aspects of Morello software work

CHERI Technical Workshop 2022

Yury Khrustalev -- Arm

06/09/2022

Agenda

- Morello toolchains and libraries: GCC, LLVM, Musl libc, etc
- OS bring-up: Android and Linux
- Morello compartmentalisation
- Tools and resources: CHERISeed, Morello IE

Software tools for Morello

GNU toolchain

- Binutils
- GCC
- GDB
- Binutils

LLVM toolchain

- Clang and tools
- LLD
- LLDB
- Runtime libraries

Morello Firmware

- Images for Morello board

Morello Platform Model

- Functional model of the Morello platform
- Supports the same software stacks as the hardware

CHERIseed

- Software-only implementation of CHERI C/C++ semantics
- Based on compiler instrumentation

Morello Instruction Emulator

- Run Morello applications
- Trace and debug
- Analyse performance

Bionic

- C library for Android

Glibc

- Implementation in progress

Musl C library

- Purecap target
- POSIX threads
- LDSO

Newlib

- C library for Baremetal

Libshim

- Syscall translation layer

Linux Kernel

- Pure Capability user ABI

Morello Development Studio

- Arm Debugger for bare-metal debug of the Morello FVP and Morello Board
- Support to import LLVM and GNU compilation tools
- Example projects to get started

Morello Firmware

- Ongoing development with quarterly feature updates
- Prebuilt images for Morello board
- Standard 64-bit components supporting booting capability-enlightened environments
- UEFI boot manager to boot OS images from PCI devices and network boot
- Morello is supported as an open research platform

Morello Android

- A full Android R profile 64-bit stack is supported on the Morello board
- Includes Morello purecap ports of the Android C library (Bionic) and a range of utilities and workloads introduced into a 64-bit AOSP environment
- Minimally modified ACK & userspace libshim support syscalls from purecap applications
- Enables a range of porting and research activities ahead of a full PCuABI kernel implementation

Morello Android

Current internal activities

- Ongoing Bionic development, aligning with evolving Kernel PCuABI

Future work areas

- Currently investigating possible activities in other areas of Android
- Examples under discussion include minimal ART configurations or constrained approaches to porting elements of the Android graphics stack

Morello Linux Kernel

- Main goal: supporting the pure-capability kernel-user ABI (PCuABI)
  - Ongoing effort to specify this ABI, candidate spec now public [1]
- Hybrid approach: in-kernel ABI unchanged, (only) user pointers become capabilities
  - Support for the traditional AArch64 ABI preserved through the COMPAT layer
- Status: prototype publicly released, targeting a transitional ABI [2]
  - Tested on the Morello board and FVP
  - Only functional PCuABI support at this stage: no enforcement of capability metadata
  - Support for a reduced set of syscalls and CONFIG options

[1] https://git.morello-project.org/morello/kernel/linux/-/wikis/Morello-pure-capability-kernel-user-Linux-ABI-specification

[2] https://git.morello-project.org/morello/kernel/linux/-/wikis/Transitional-Morello-pure-capability-kernel-user-Linux-ABI-specification

Morello Linux Kernel

- Current focus: kernel testing
  - Morello kselftests run in Morello GitLab CI
  - A subset of LTP and Musl tests are being deployed
- Next steps:
  - Progressing towards the complete PCuABI, primarily by restricting and enforcing capability bounds / permissions
  - Extending the number of supported syscalls and CONFIG options
- All development is now happening in the open - contributions welcome!
  - Kernel repo: https://git.morello-project.org/morello/kernel/linux
  - Mailing list: https://op-lists.linaro.org/mailman3/lists/linux-morello.op-lists.linaro.org/

Morello toolchains

GNU toolchain

- Targeting Baremetal and Linux
- Binutils, GDB and other tools
- Glibc

- git://sourceware.org/git/binutils-gdb.git
  branch: users/ARM/morello-binutils-gdb-master
- git://gcc.gnu.org/git/gcc.git
  branch: vendors/ARM/heads/morello
- git://sourceware.org/git/glibc.git
  branch: arm/morello/main
- git://sourceware.org/git/binutils-gdb.git
  branch: users/ARM/morello-binutils-gdb-master

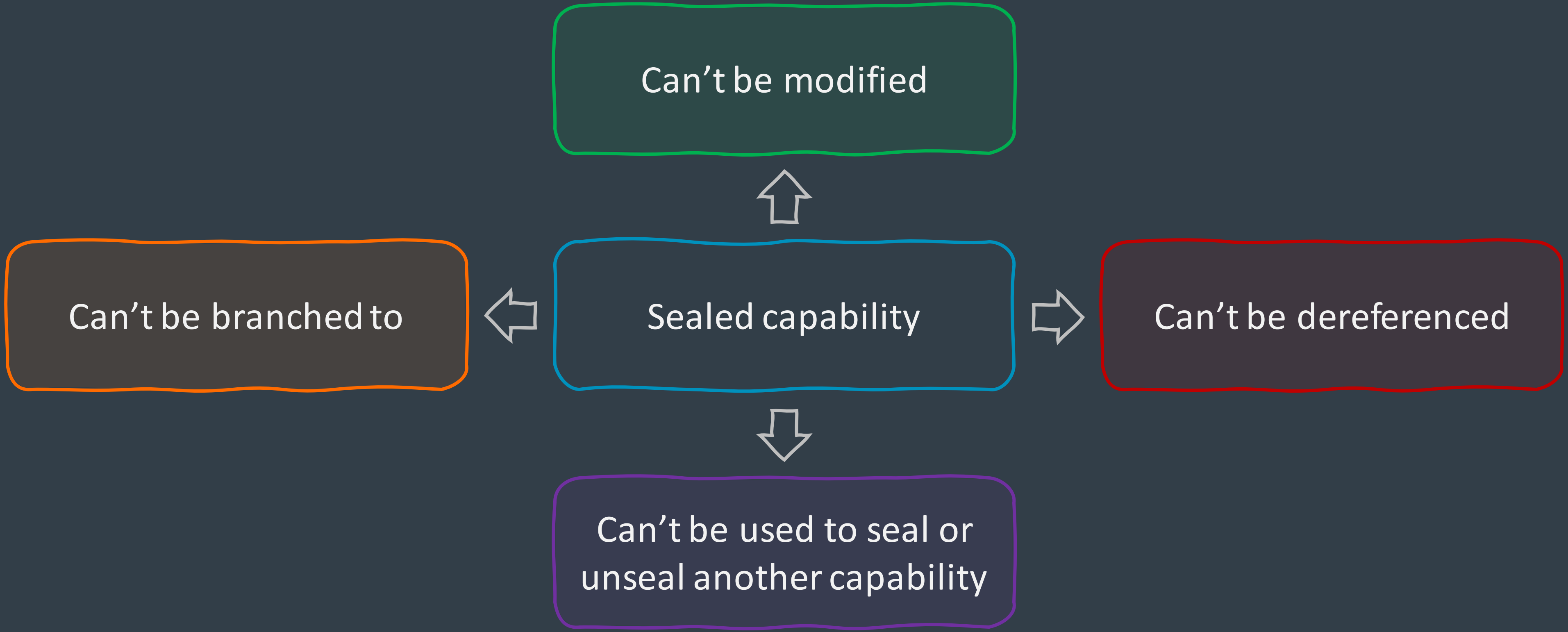
LLVM toolchain

- Targeting Baremetal, Android and Linux
- LLDB and other usual LLVM utilities

- https://git.morello-project.org/morello/llvm-project
- https://git.morello-project.org/morello/llvm-project-releases
  branch: morello/release-1.4
  branch: morello/baremetal-release-1.4
  branch: morello/linux-aarch64-release-1.4

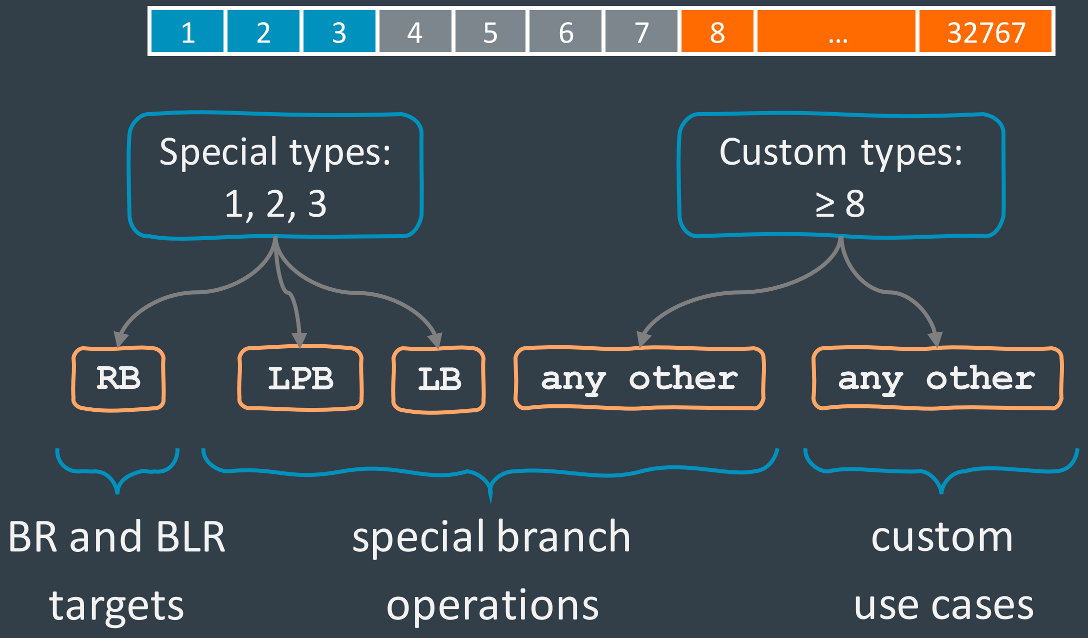
Compartments

Sealed Capabilities in Morello



Sealed Capabilities in Morello

Object Type (15 bits)



To seal a capability, you need

- Sealing with **any** object type:
  - an unsealed capability
  - **sealer** capability that
    - is in bounds
    - has SEAL permission
    - is valid (tag == 1)
    - has value == object type
- Sealing with reserved object type:
  - an unsealed capability

Compartmentalisation

- Consider compartmentalisation in terms of transitions across boundaries between security domains
- Symmetrical model that works for any type of trust-mistrust relationship (both ways)
- Focus on the information that is transferred across the boundary in any direction
- …and what data is protected against leaking through the boundary

- **Compartmentalisation is managing transitions between security domains**

Threat model

Overview

- An application is partitioned into security domains
- Some security domain may be compromised
- Aim is to prevent escaping from the compromised compartment
- We want to restrict access to data and execution of code in other security domains

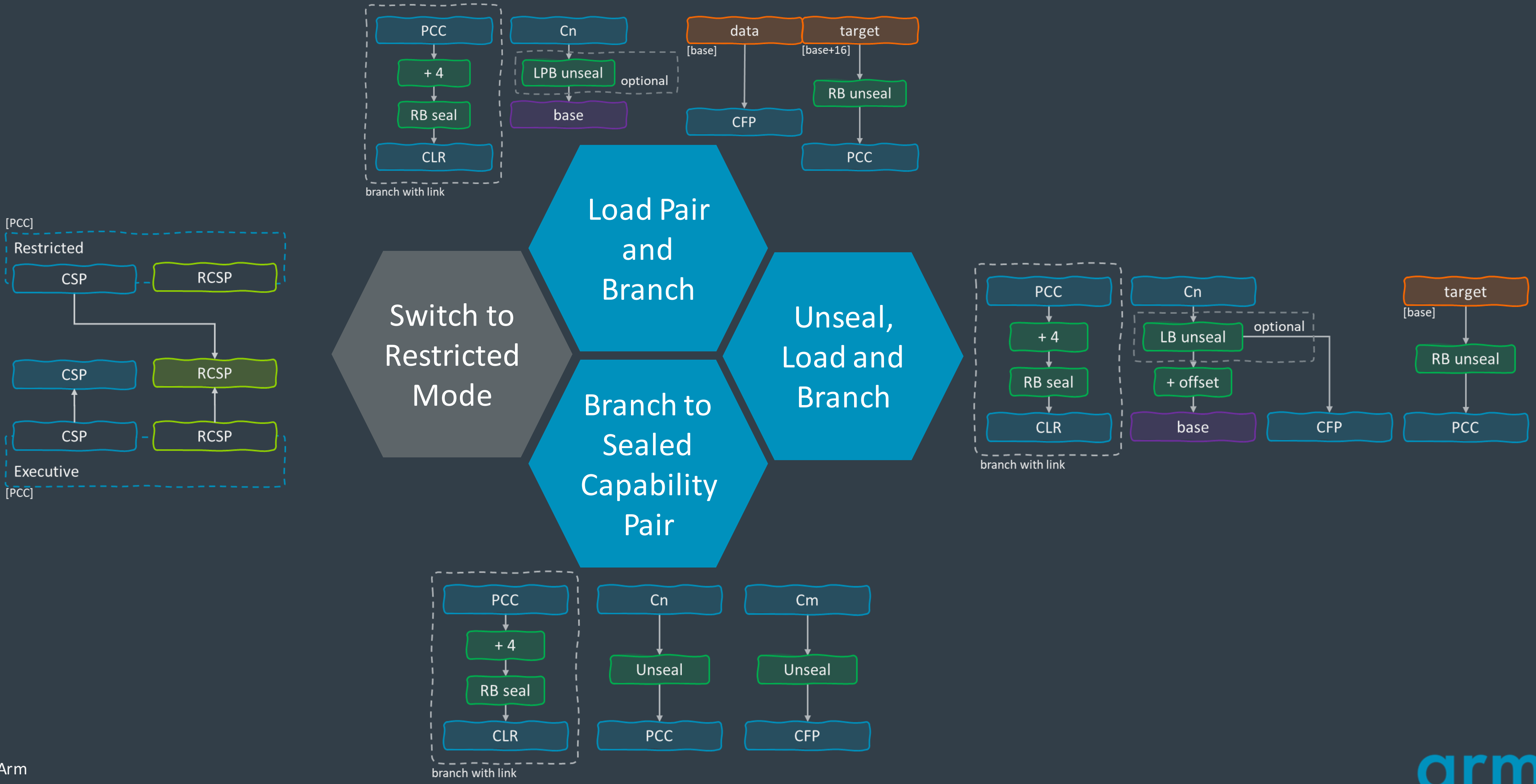
An adversary has…

- Arbitrary* read and write memory gadgets
- Arbitrary read and write register gadgets
- Forged code (function) pointers
- Sealing and unsealing gadgets
- CVTP gadget (makes arbitrary code pointers)

An adversary can…

- Read and write data inside the compartment
- Access and infer data in the compartment
- Modify register values to "program" gadgets
- Create valid code pointers
- Take over control flow*

Primitives for cross-domain transition



Primitives for cross-domain transition

LPB, ULB, BSP

- Work on capability pair `{data, code}` that can be referred to as "compartment descriptor"
- Are associated with a branch operation
- Update frame pointer CFP
- Involve an unsealing-during-branch operation which means they work on sentries
- Are easy to deploy
- Non-atomic stack isolation

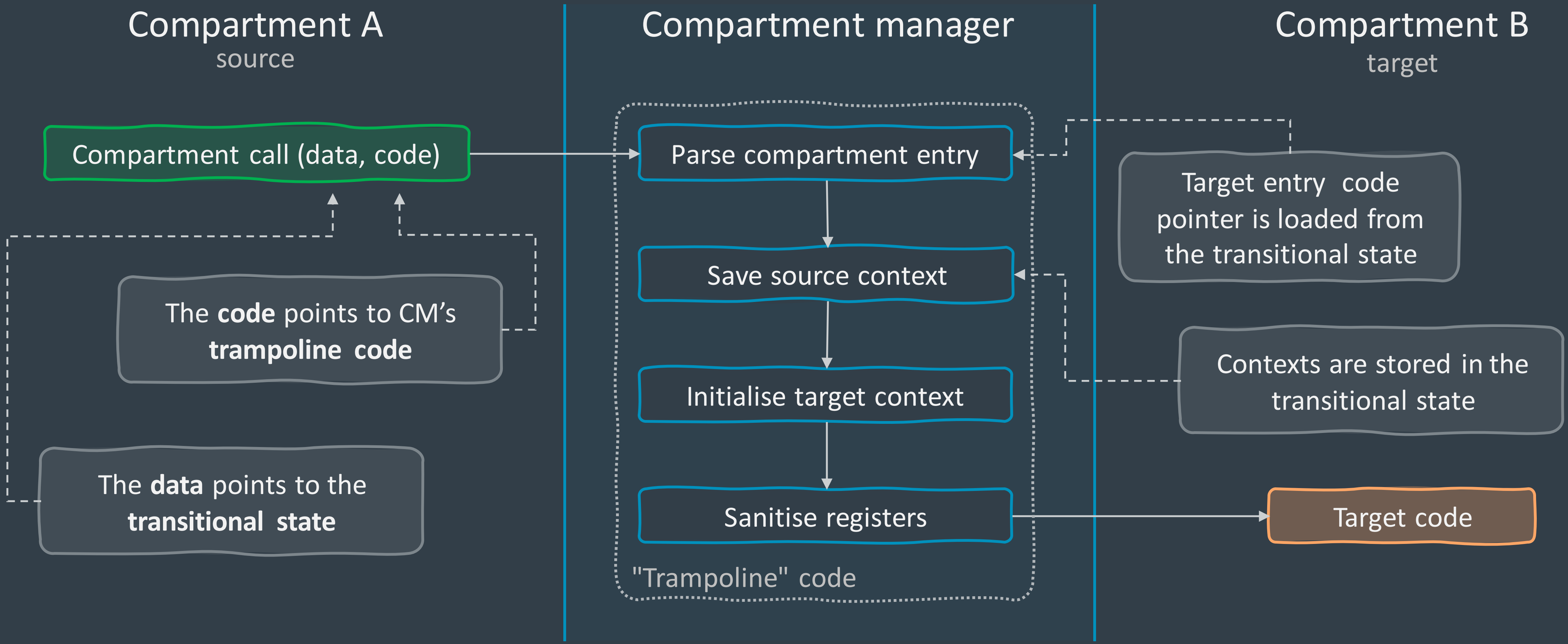
- Have similar security properties, however Branch to Sealed Pair primitive is robust against compartment descriptor forgery attack

Switch to Restricted Mode

- Switch to Restricted Mode is special
- No custom branch operation is required
- No unsealing (except for common RB-unsealing).
- Requires special version of libc and dynamic linker
- A special version of kernel supporting this might also be handy, although not required
- Atomic stack isolation

Morello Compartmentalisation

Very basic compartment switch idea



Compartment A
source
Compartment manager
Compartment B
target
Compartment call (data, code)
Parse compartment entry
Target entry code pointer is loaded from the transitional state
Save source context
The **code** points to CM's **trampoline code**
Contexts are stored in the transitional state
Initialise target context
The **data** points to the transitional state
Sanitise registers
Target code
"Trampoline" code

Purecap Compartmentalisation

- Restricting stack accesses by initializing compartment's stack and switching stacks
- Heap accesses protected by capabilities returned by `mmap` or `malloc` or alike
- PC-relative accesses may be restricted in terms of PCC bounds and permissions (for example, preventing access to sensitive system registers)
- Any other sources of capabilities should be sanitized

- Arbitrary sized compartments: function, object, DSO, etc.
- Reusable compartment allocations (depending on trust model)

Trying out Morello at home

How to try Morello

- GNU and LLVM toolchains for Morello targets are available in source and binary form
- Arm traditionally provides Morello FVP that can be used to boot Android or Linux
- This can be used to build and run purecap Morello applications

- There are other tools that could be of interest for researchers and software developers
- All the above aims to provide quick hands-on experience with CHERI / Morello to help understand new properties, security guarantees, performance and deployment applications

CHERIseed and libshim

- CHERIseed is based on CHERI/Morello LLVM and uses sanitizer-like approach
- Allows to build programs with CHERI semantics on widely available machines
- Enables step-by-step approach for porting existing code to CHERI
- Provides CHERI at C/C++ level and helps understand capabilities
- Open source

- Start here:
  - https://git.morello-project.org/morello/llvm-project/-/blob/cheriseed/clang/docs/CHERIseed.rst
  - https://git.morello-project.org/morello/llvm-project/-/blob/cheriseed/clang/docs/CHERIseedDesign.rst

Morello Instruction Emulator

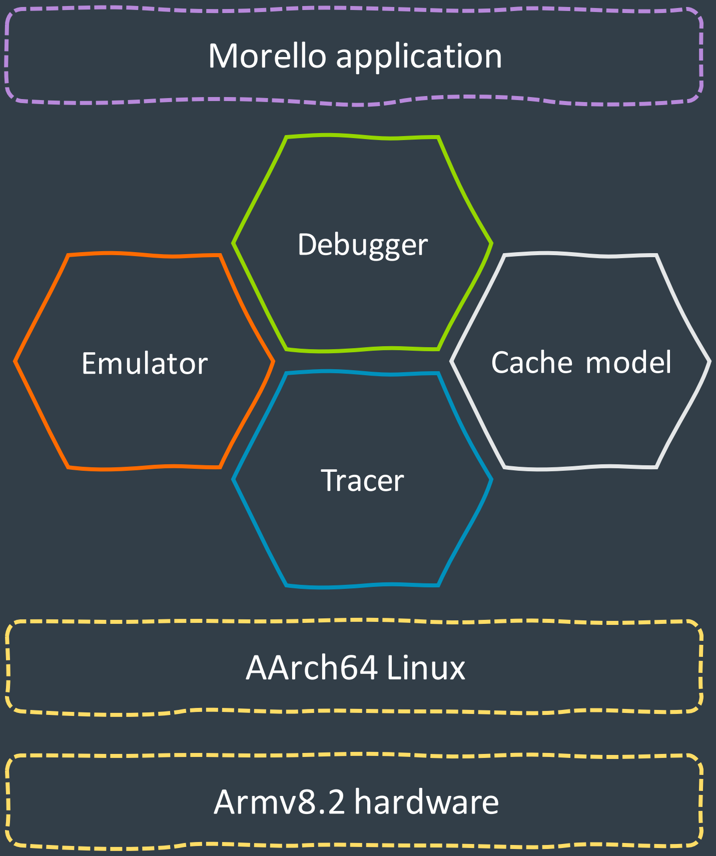
What's inside

- DBT tool for running Morello applications in a non-Morello Linux environment
- Capability registers
- Memory tags
- Syscall translation: libshim or work-in-progress built-in support

What it does

- Experiments with Morello applications on non-Morello AArch64 Linux systems
- Evaluate compartment solutions and experiment with the new syscall ABI
- Test, debug and trace existing software being ported to Morello
- Trace-based performance analysis and cache modelling

How it works



https://developer.arm.com/downloads/-/morello-instruction-emulator
https://developer.arm.com/documentation/102270/

Resources and more information

- https://www.arm.com/architecture/cpu/morello
- https://developer.arm.com/Tools and Software/Morello Development Tools
- https://www.morello-project.org/

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה