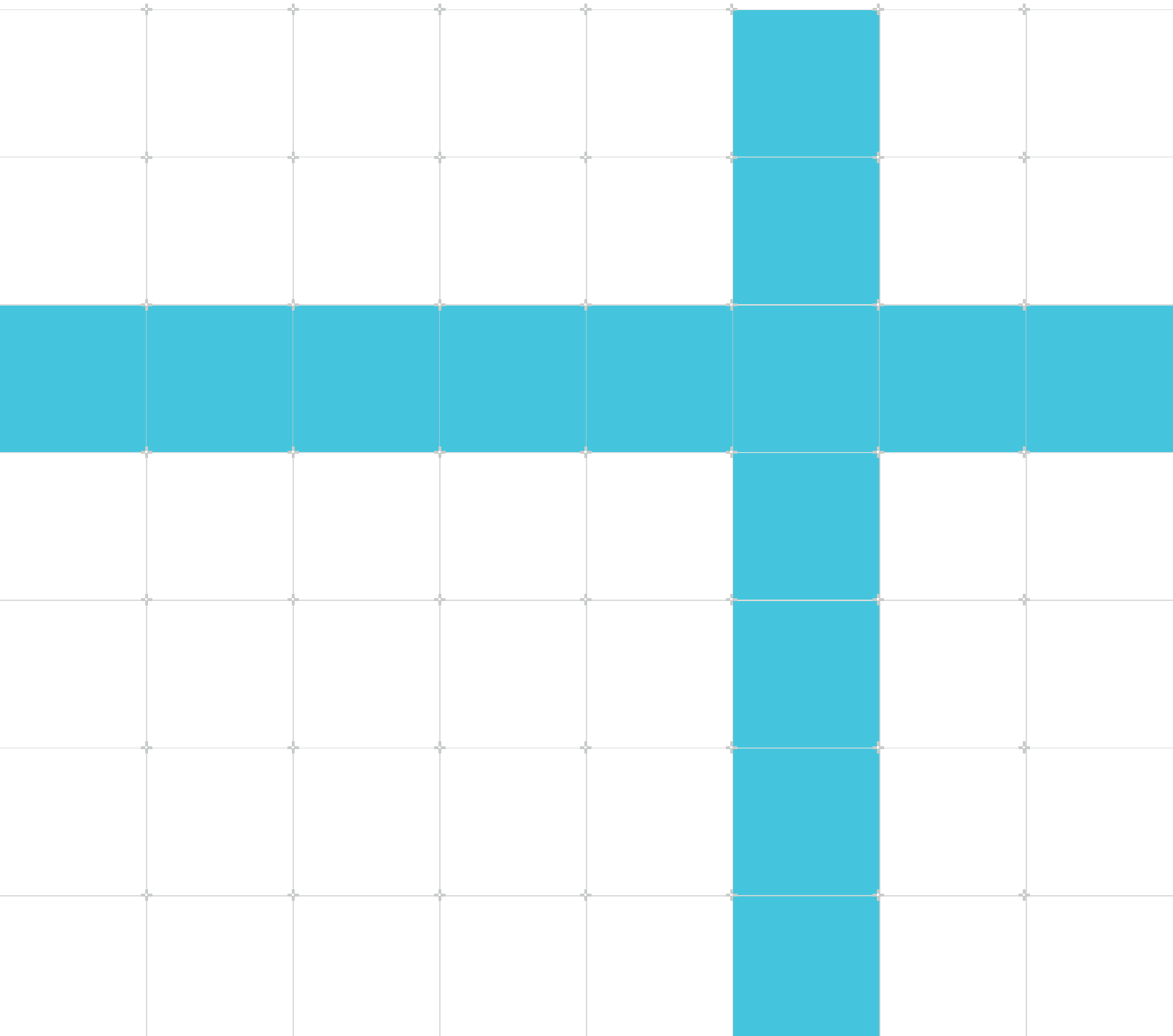
Morello Prototype Architecture Overview

Version 1.0

Issue
den0133_0100_en



Morello Prototype Architecture Overview

Release information

Document history

Issue	Date	Confidentiality	Change
0100	28 February 2022	Non-Confidential	Initial release

Proprietary Notice

or indirectly, in violation of such export laws. Use of the word "partner" in reference to Arm's customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

Arm Limited. Company 02557590 registered in England.

110 Fulbourn Road, Cambridge, England CB1 9NJ.

(LES-PRE-20349)

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Unrestricted Access is an Arm internal classification.

Product Status

The information in this document is Final, that is for a developed product.

Feedback

Arm® welcomes feedback on this product and its documentation. To provide feedback on the product, create a ticket on https://support.developer.arm.com

To provide feedback on the document, fill the following survey: https://developer.arm.com/documentation-feedback-survey.

Inclusive language commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used language that can be offensive. Arm strives to lead the industry and create change.

We believe that this document contains no offensive language. To report offensive language in this document, email terms@arm.com.

Contents

1 Overview **6**
1.1 The Morello Program approach 7

2 Morello prototype architecture **8**
2.1 Pointers and capabilities 9
2.1.1 Pointer protection in the Morello architecture 11
2.2 Security compartments 11
2.2.1 Morello compartments 12

3 Arm Morello implementation **14**
3.1 Instruction Set Architecture 14
3.2 Capability registers 14

4 Tools and software ecosystem for Morello **15**
4.1 Bare-metal enablement 15
4.2 Android enablement 16
4.3 Software development environment 16
4.4 Compilation modes 17

5 Summary and next steps **18**
5.1 Morello Platform Model availability 18
5.2 Additional information 18

1 Overview

Security is one of the top priorities in designing a system using microprocessors. Security remains a top priority across all computing platforms irrespective of the target application whether designing:

- Server-based system
- Client-based mobile platform
- Automotive system
- Medical equipment

Designing a system that is completely immune to malware is practically impossible. There are regular reports of cybercrime attacks on major businesses in the media. These attacks are costly and it is a constant battle to defend against them. Here are just a few cybercrime statistics:

Figure 1-1: Summary of cybercrime statistics

The approach to building a robust, Secure system has been to track the most common vulnerabilities and to ensure security flaws are resolved in the new design. This approach includes using dedicated in-house teams to identify possible flaws. Secure software solutions and system design guidelines have evolved based on security issues and are likely to evolve further.

However, not all security issues can be countered by following Secure system design and software guidelines alone. This is a time-consuming and expensive exercise, and therefore warrants investigating the underlying hardware architecture to address large-scale and repetitive issues.

For example, memory vulnerability issues due to pointer corruption are a known problem. Still, there are no systems that are completely immune to pointer corruption. Arguably, a hardware solution to protect pointers might help solve these kinds of multiple memory vulnerability issues.

A more effective approach to building a robust secure system is to implement an architectural solution to counter security vulnerability. However, architectural solutions face other issues, including the following:

- New architecture design is time consuming.
- Validating a new design involves significant time and effort.
- Existing software may not be compatible.

- The complete ecosystem, including software, compilation tools, operating systems, and debug tools may need to be rebuilt.

1.1 The Morello Program approach

Arm is leading a research program called Morello, working with key partners to evaluate this approach, including:

- University of Cambridge
- University of Edinburgh
- Linaro

The research is funded by the UK Research and Innovation. The Morello Program uses new hardware mechanisms to access memory, based on the Cambridge University CHERI research architecture, that should mitigate a high proportion of memory vulnerability issues.

Morello has the potential to change radically the way processors are designed and programmed in the future, enabling improved security to be built into the hardware. This is a major step change in architecture, and the associated software and tooling. Therefore, the Morello program's objective is to create and evaluate this new approach over the long term.

The design goals of the Morello prototype architecture are:

- Fine-grained memory protection, leading to increased memory safety
- Scalable compartmentalization and isolation
- Defensive execution technologies

The next section describes the Morello prototype architecture and features, while highlighting the protection offered.

2 Morello prototype architecture

Since 2014, Arm has been collaborating with the University of Cambridge and SRI international in the development of the Capability Hardware Enhanced RISC Instructions (CHERI) architectures. Morello is a prototype architecture that adapts the hardware concepts of CHERI into the Arm architecture, aiming to improve the robustness and security of systems.

The Morello architecture extends the Armv8-A AArch64 state with the principles proposed in version eight of the CHERI Instruction Set Architecture (ISA). Morello supports a new data type called a capability.

A capability is a 129-bit quantity comprising elements such as address, allowable address range (bounds) information, and permission information. Capabilities are held in registers or memory and can be used as code and data pointers:

- Code capability

  Code capability refers to a capability intended to be used as a code pointer, whose capability permissions have been configured to permit instruction fetch, for example execute rights. Typically, write permission not granted through an executable capability, in contrast to a data capability. Code capabilities are used to implement control-flow robustness by constraining the available branch and jump targets.
- Data capability

  Data capability refers to a capability whose capability permissions have been configured to permit data load and store, but not instruction fetch rights, in contrast to a code capability.

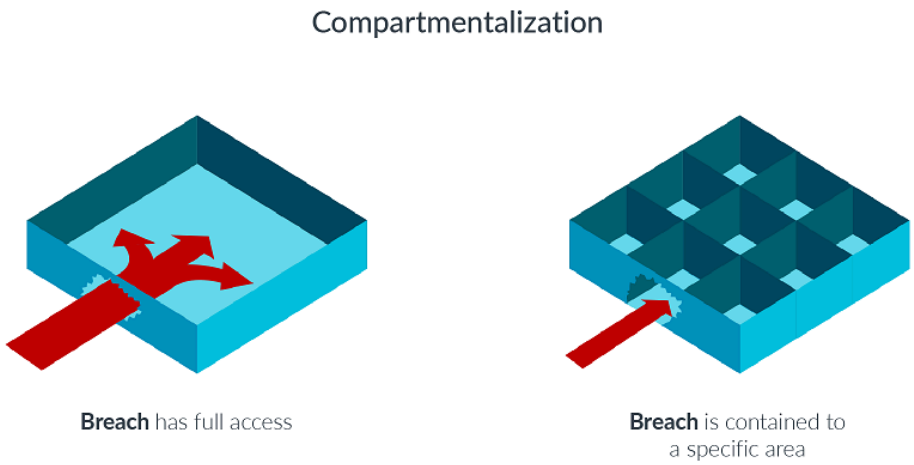
For example, when a load or store operation is performed using a capability, the address in the capability is checked against the bounds.This check happens to validate whether access to the memory location is permitted. There are permission bits to control the type of access, for example read-only or read/write. The bounds and permission check enables the capability to limit how pointers can be used by scoping the ranges of memory (bounds) and operations that can be performed (permissions).

Replacing or augmenting pointers with capabilities in a program improves memory safety, which is a key step for security.

The benefit of capability hardware technology goes beyond memory safety. Capabilities can be used as a building block for the finer-grained compartmentalization of software, as explained in Security compartments. Software that is constructed with fine-grained compartmentalization could be inherently more robust and resistant to attacks.

A powerful feature of compartmentalization is that, if an attacker compromises a compartment, the attacker cannot easily break out of the compartment. This feature makes harder to access any other information, or to take overall control of the computing system. The following diagram shows the advantage of compartmentalization:

Figure 2-1: Advantage of compartmentalization



The next section details the concept of capabilities in the Morello prototype architecture and shows how they are used to implement fine-grained memory protection and secure compartments.

2.1 Pointers and capabilities

The Morello protection model is primarily a change to the way memory is accessed, using rich pointers that can define constraints for memory access.

In the Armv8-A architecture, a pointer is held in a 64-bit general-purpose register. In the Morello prototype architecture, a pointer is held in a 129-bit capability register. Along with the address of the virtual memory to be accessed, the Morello capability includes additional metadata:

- The virtual address range that can be accessed (referred to as bounds)
- The access permissions (read, write, or execute)
- An object type that indicates whether a capability is either sealed or unsealed
- A capability tag-bit indicating whether the capability is valid or not

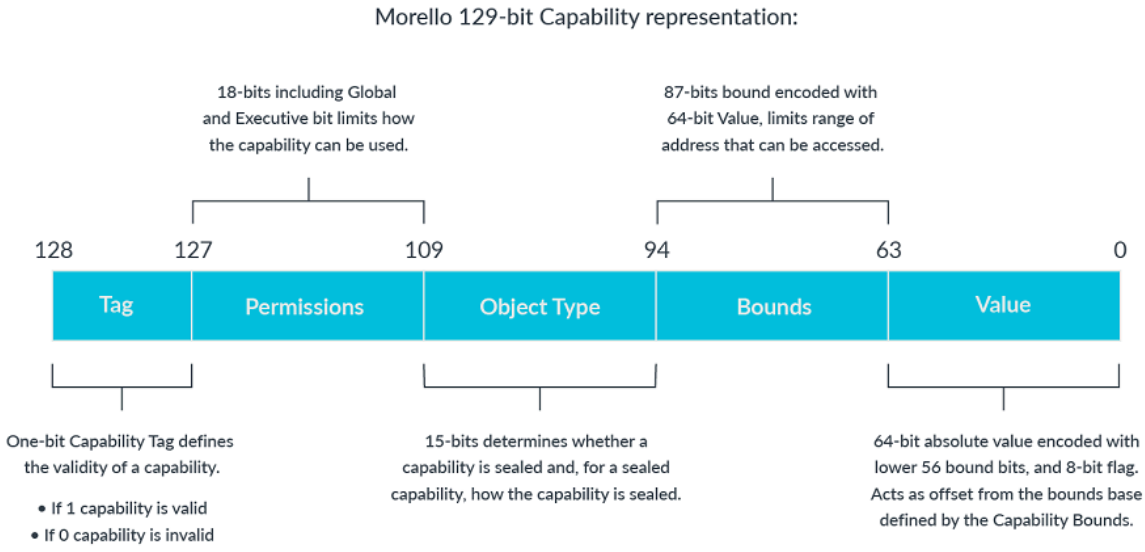
Every memory access arising from the processor must check access against a capability. These are:

- Load instructions
- Store instructions
- Instruction fetch using the Program Counter (PC)

If the capability protection checks pass, the memory access instruction executes successfully, or an exception is generated.

The following diagram shows the bit field representation of a Morello 129-bit capability:

Figure 2-2: Morello 129-bit capability representation



The Morello architecture supports 31 individual 129-bit general-purpose registers, C0 to C30, that can hold capabilities, and these registers can be used as pointers while performing memory accesses. Capabilities can only be loaded from or stored into memory using the new access instructions. For information about the new instructions, please refer to the Arm Architecture Reference Manual Supplement Morello for A-profile Architecture documentation.

For every 16 bytes of memory, Morello defines an associated memory tag-bit that is atomically updated on any store to that memory. When a capability is stored to memory, 128 bits of the capability are stored to the 16 bytes and the capability tag-bit is stored to the associated memory tag.

The software can't address the tag-bit in the memory. When reading the capability from memory using a load capability instruction, the associated memory tag is atomically copied into the 129th bit of the capability register.

The tag-bit indicates whether the capability is valid:

- Valid capability: tag-bit is set (1)
- Invalid capability: tag-bit is clear (0)

Any non-capability memory write operation clears a tag-bit. This detects any manipulations to the capability while capabilities are saved in memory. A subsequent attempt to use an invalid capability generates an exception. When using capabilities as pointers, the capability tag check helps protect pointer integrity. The next section highlights some protection properties that capabilities grant to pointers.

2.1.1 Pointer protection in the Morello architecture

There are several methods of data pointer protection:

- Pointer integrity protection: Overwriting the pointer value within a capability in memory not allows construction of a usable pointer.
- Pointer provenance checking and monotonicity: Pointers must be derived from prior pointers through manipulations that cannot increase the range or permissions of the pointer.
- Bounds checking: Pointers cannot be moved outside of their allocated range while being used for load, store, or instruction fetch.
- Permissions checking: Pointers cannot be used for a purpose not granted by permissions. This prevents data pointers from being used for code execution.
- Bounds or permissions subsetting: Programmers can explicitly reduce the rights associated with a capability, for example by further limiting its valid range, or by reducing permissions to perform load and store operations. This might be used to narrow ranges to specific elements in a data structure or array, such as a string within a larger structure.

  In addition, Morello capabilities can prevent code pointers from being corrupted or misused. This can limit various forms of control-flow attacks, including overwriting of return addresses on the stack, and pointer reuse attacks such as Return-Oriented Programming (ROP) and Jump-Oriented Programming (JOP).
- Return-address protection: Modifying the procedure return addresses stored in the stack is a well-known attack used by malware and known as ROP. This vulnerability can be mitigated by storing a capability for the procedure return address. Any attempt to modify the parts of the capability in memory invalidates the corresponding memory tag, rendering the capability unusable.
- Function-pointer protection: Modifying the branch target addresses stored in memory is a well-known attack used by malware and is known as JOP. This vulnerability can be mitigated by storing a capability for the branch target address. Any attempt to modify the parts of the capability stored in memory invalidate the tag-bit, rendering the capability unusable.

2.2 Security compartments

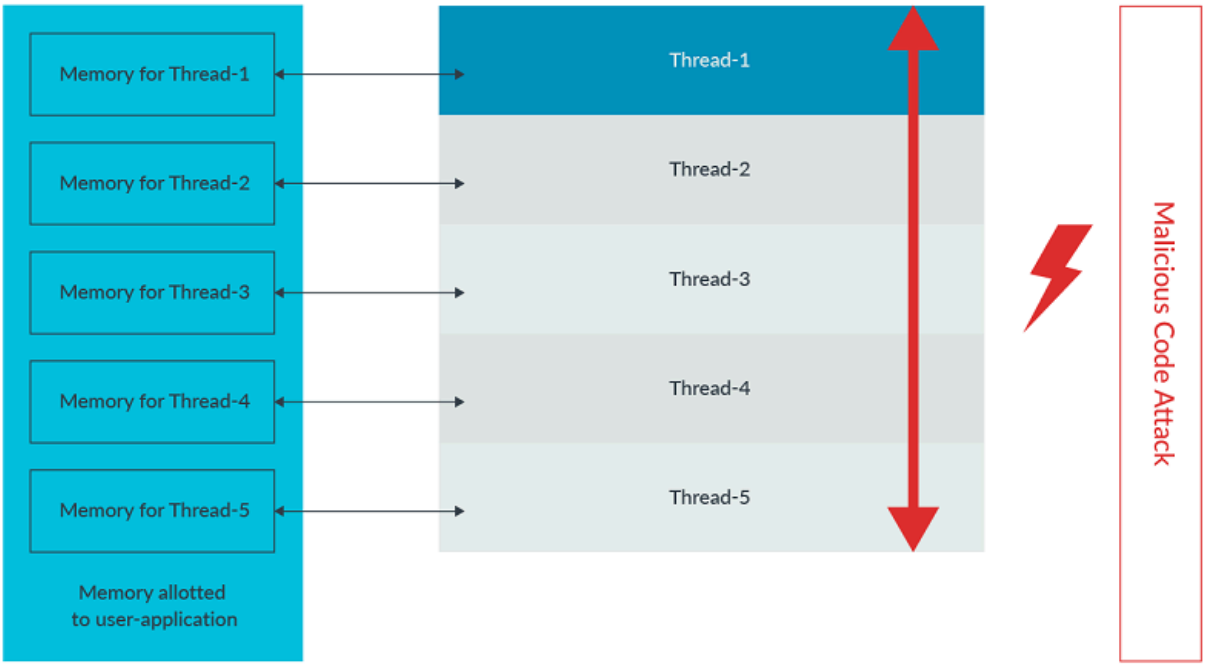
Software compartmentalization is one of the few techniques that can mitigate future unknown classes of software vulnerability and exploitation. The software compartmentalization protection properties do not depend on the specific vulnerability or exploit class used by the attacker.

In software compartmentalization, complex bodies of software - for example, a user application - are broken into multiple components that run in isolation from one another. Different components interact with each other in an explicitly defined interface. This section provides an abstract of software compartments in general and explains how isolation of these software compartments can be improved using Morello capabilities.

One of the key features of an operating system is to manage the isolation of memory used by user applications. This is achieved using the Memory Management Unit (MMU). The MMU, under control of the operating system, manages memory for user applications at the page granularity.

A user application can further divide its functionality into multiple threads within the application space, and there is scope to run these threads in isolated software compartments. Each thread managed by the user-application has access to all the virtual memory allocated to the application space by the kernel. Though each thread is intended to run in isolation, the application virtual memory is vulnerable to software bugs and common memory attacks. Therefore, any attack to one of the threads by malicious code increases the attack surface to all software compartments, as shown in the following diagram:

Figure 2-3: Malicious code attack to one compartment increases the surface attack across all the compartments as depicted by the red arrow.



2.2.1 Morello compartments

The construction of software compartments using Morello capabilities helps mitigate some known vulnerabilities.

Fine-grained memory protection

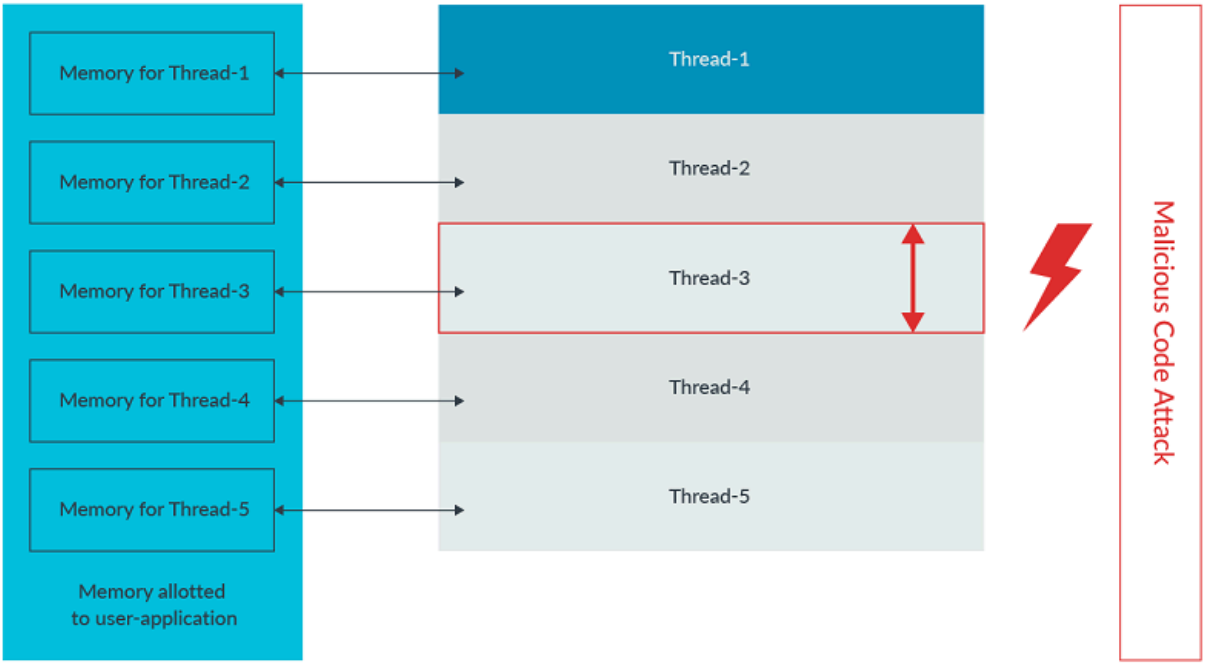
Fine-grained memory protection is the description of available code and data. In these, the capability bounds and capability permissions are made as small as possible to limit the potential effects of software bugs and vulnerabilities. This approach can be applied both to code capabilities and data capabilities. This approach also offers effective vulnerability mitigation through techniques

such as control-flow robustness, as well as supporting higher-level mitigation techniques for software compartmentalization.

Using Morello capabilities for software compartments

Effective configuration of code capabilities, data capabilities, and fine-grained memory protection, helps isolate software components from one another, enabling vulnerability mitigation at the user space for software compartments. The following diagram shows the potential attack surface in the case of malicious-code attack into Thread-3 using appropriate Morello capabilities for software compartments:

Figure 2-4: Malicious code attack to Morello software compartment constructed using capabilities and memory tagging restricts the surface attack to only one compartment as depicted by the red arrow.



Note

In this example, we illustrate each thread in the user application as an isolated software compartment. This can be further extended so that there are multiple isolated software compartments within a single thread.

3 Arm Morello implementation

The Morello architecture is implemented in the Morello Program as an experimental extension to the Armv8-A architecture version Armv8.2-A.

The Morello architecture:

- Extends the Armv8.2-A profile with 129-bit capabilities
- Is only supported in the AArch64 state
- Is backwards compatible with, and complementary to, the existing Armv8-A architecture

The Morello architecture does not support the following:

- The AArch32 state
- Mixed-endian support at any Exception level
- Fixed big-endian. The architecture only supports fixed little-endian.

An abstract of key changes and features in the Morello architecture to support capabilities is highlighted in this section.

3.1 Instruction Set Architecture

The A64 ISA is extended with instructions to manipulate, copy, and retrieve fields from capabilities. A variant of the A64 ISA, C64, is added to provide a richer set of instructions to use capabilities.

3.2 Capability registers

The Morello architecture introduces the term capability register to define a register that can hold a capability. Capability registers are 129 bits. The general-purpose registers are extended to hold capabilities. Capability registers have the following access views:

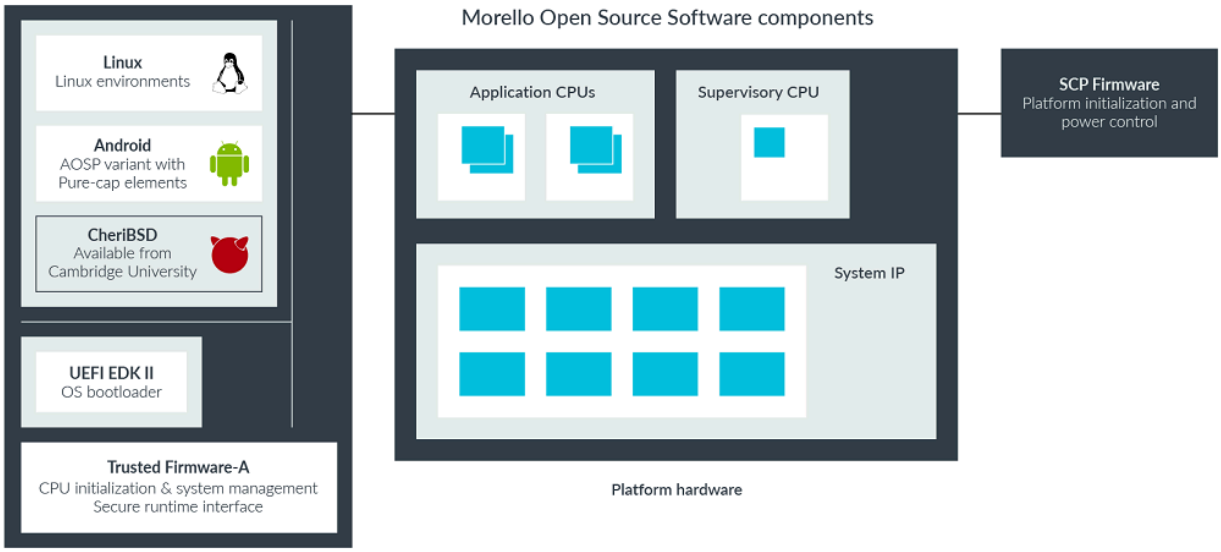
- 129-bit when referenced as Cn [C0 to C30]
- 64-bit when referenced as Xn [X0 to X30]
- 32-bit when referenced as Wn [W0 to W30]

For a list of the complete features supported in the Morello implementation, please refer to the Morello Specification.

4 Tools and software ecosystem for Morello

The following diagram shows a high-level view of the software stacks targeting the Morello hardware and Fixed Virtual Platforms (FVP):

Figure 4-1: Software stack for Morello



These stacks and the supporting tooling are intended to provide a foundation for ecosystem research. This foundation enables collaboration on existing work packages and new work on alternate RTOS/OS environments, tools, and workloads. The functionality of these software components evolves in stages throughout the lifetime of the Morello Program. Integrated stack releases (manifests, build scripts, and documentation) and associated component forks, are available from Morello Git group. More information is also available at Morello Open Source Software page.

4.1 Bare-metal enablement

The exit from the firmware stack supports Bare-metal development at two points:

- Post System Control Processor (SCP) execution: System level IP is initialized. Development is possible from application processor reset. SCP execution supports true bare-metal scenarios.
- Post Trusted Firmware-A (TF-A) execution: The lead application processor is initialized and runtime services are available. TF-A execution supports ports of new RTOS environments and more complex bare-metal workloads.

For more information, refer to `standalone-baremetal-readme.rst` in the documentation repository.

4.2 Android enablement

An evolving Android environment has been available for Morello since the first release in October 2020.

This includes a minimal headless (without user interface) system Android (64-bit) profile suitable for use with the FVP. Full Android boot is supported on the Morello hardware platform.

Support for pure capability (purecap) applications along with several example ports is provided by a Morello Android Common Kernel(ACK) and Bionic library variants built using the CHERI LLVM/ Clang toolchain.

For more information on the status of the Android environment, refer to `android-readme.rst` in the documentation repository.

4.3 Software development environment

LLVM open-source toolchains supports Morello based on the CHERI Clang/LLVM toolchain from the University of Cambridge (UoC). LLVM compiler with Morello support is available.

The toolchain provides:

- A C/C++ compiler: `clang`
- Linkers: `lld`
- A debugger: `lldb`
- Various utilities, such as an assembler and disassembler

The toolchain is derived from Clang, and all Clang driver options remain valid. Refer to the Clang Compiler User's Manual for further details.

Depending on the target system, use the appropriate target option:

- `-target aarch64-linux-android` for Android
- `-target aarch64-unknown-freebsd12.0` for CheriBSD
- Cross-compiling using the toolchain needs additional arguments. For more information, refer to Cross-compilation using Clang.
- Use `--sysroot` to indicate a sysroot for your target
- Use `--fuse-ld=lld` on targets where `lld` is not the default linker

4.4 Compilation modes

The toolchain can generate code for three different modes:

- Plain AArch64. All binaries can be run without modification on Morello but will not use the benefits of capability.
- Capability-enabled AArch64. Hybrid, with an ABI backwards compatible with the AArch64 ABI.
- Pure Capability ABI.

Code generation for the Pure Capability ABI is restricted to the C64 instruction set. For the legacy AArch64 ABI, code generation must use the A64 instruction set.

5 Summary and next steps

As Morello is a research project, the intent is to evaluate various usage of capabilities on a real hardware platform. This evaluation helps determine if CHERI and its capability technologies fulfill the promise of making processors more Secure. The next stage is to test the architecture through the Morello evaluation boards, which will be distributed to a broad set of industry and academic partners. The results of this investigation will feed back and influence whether Morello is incorporated into future Arm architecture designs.

The Morello System Development Platform (SDP) is a prototype development board that contains a Morello System on Chip (SoC). This is the only physical implementation of the Morello prototype architecture.

The intent of the Morello board is to:

- Experiment with the various uses of capabilities
- Test and validate the performance impact of compartmentalization using capabilities against existing software compartments
- Investigate complex usage models on an actual hardware system
- Formally validate the various protection mechanisms offered by capabilities
- Use the learning as feedback to determine if this is suitable to be implemented into the actual Arm architecture in the future

The Morello prototype board is available to the appropriate software companies, tool developers, and leading academic institutions. Any interested parties must register at the Digital Security by Design page.

5.1 Morello Platform Model availability

Delivery of the Morello evaluation board is targeted for Q1 2022. Prior to the launch of the Morello board, a Fixed Virtual Platform (FVP) has been developed, known as the Morello Platform Model. As access to the Morello board will be limited, those without access can use the FVP to develop and run the Morello software.

The Morello Platform Model model uses Arm technology to create a virtual model of the system hardware available to use in a development environment. This simulator, including the toolchain, software, and documentation, will allow Morello researchers and DSbD technology-based providers to begin writing code and running software. The Arm Morello Platform Model does not require a license, runs on a Linux host machine, and is freely available from the Arm Developer website.

5.2 Additional information

Material from the Morello specification and Technical Report Number 927, Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture, has been extracted and used in this

overview. Additional information published by Arm and other third parties is listed in the following table:

Document name	Document ID	Licensee only
Arm Architecture Reference Manual Supplement - Morello for A-profile Architecture	ARM DDI0606	No
Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture Version 8	UCAM-CL-TR-951	No
An Introduction to CHERI	UCAM-CL-TR-941	No