CHERI: Architectural support for strong memory safety and scalable compartmentalized software

Robert N. M. Watson, Simon W. Moore, Peter Sewell, Peter G. Neumann,
Brooks Davis

Hesham Almatary, Ricardo de Oliveira Almeida, Jonathan Anderson, Alasdair Armstrong, Rosie Baish, Peter Blandford-Baker, John Baldwin, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, Brian Campbell, David Chisnall, Jessica Clarke, Nirav Dave, Lawrence Esswood, Nathaniel W. Filardo, Franz Fuchs, Dapeng Gao, Ivan Gomes-Ribeiro, Khilan Gudka, Brett Gutstein, Angus Hammond, Graeme Jenkinson, Alexandre Joannou, Mark Johnston, Robert Kovacsics, Ben Laurie, Ka Wing Li, Zhuo Ying Jiang Li, Jessica Man, A. Theo Markettos, J. Edward Maste, Alfredo Mazzinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, George Neville-Neil, Kyndylan Nienhuis, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Allison Randal, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Thomas Sewell, Stacey Son, Ian Stark, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Jonathan Woodruff, Hongyan Xia, Vadim Zaliva, and Bjoern A. Zeeb

SRI International Capabilities Limited University of Cambridge

Rennes – 12 September 2025

Approved for public release; distribution is unlimited. Sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237 ("CTSRD"), with additional support from FA8750-11-C-0249 ("MRC2"), HR0011-18-C-0016 ("ECATS"), FA8650-18-C-7809 ("CIFV"), HR001122C0110 ("ETC"), HR001123C0031 ("MTSS"), and FA8750-24-C-B047 ("DEC") as part of the DARPA I2O CRASH, I2O MRC, MTO SSITH, and I2O CPM research programs. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Approved for public release; distribution is unlimited.

CHERI development was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237 (“CTSRD”), with additional support from FA8750-11-C-0249 (“MRC2”), HR0011-18-C-0016 (“ECATS”), FA8650-18-C-7809 (“CIFV”), HR001122C0110 (“ETC”), HR001123C0031 (“MTSS”), and FA8750-24-C-B047 (“DEC”) as part of the DARPA I2O CRASH, I2O MRC, MTO SSITH, and I2O CPM research programs. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Arm’s Morello and portions of the Morello-enabled software stack were supported by the Innovate UK project 105694 (“Digital Security by Design (DSbD) Technology Platform Prototype”), Innovate UK project 107145 (“Assessing the Viability of an Open-Source CHERI Desktop Software Ecosystem”), Innovate UK project 10027440 (“Developing and Evaluating an Open-Source Desktop for Arm Morello”), and Innovate UK project 10027332 (“MOJO - A Robust Java Virtual Machine for Morello”).

We further acknowledge EPSRC REMS (EP/K008528/1), EPSRC CHaOS (EP/V000292/1), ERC ELVER (789108), the Isaac Newton Trust, the UK Higher Education Innovation Fund (HEIF), Thales E-Security, Microsoft Research Cambridge, Arm Limited, Google, Google DeepMind, HP Enterprise, and the Gates Cambridge Trust.

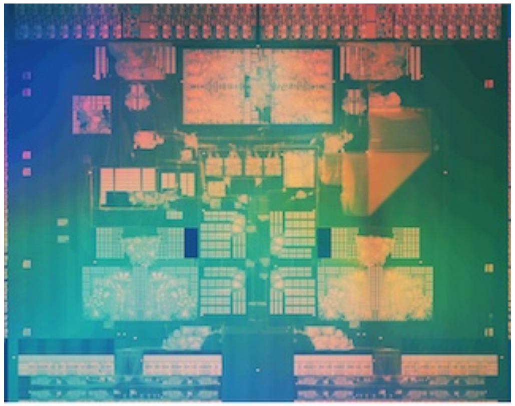
Introduction

- **CHERI is a new processor technology that mitigates software security vulnerabilities**
  - Developed by the University of Cambridge and SRI International starting in 2010, supported by DARPA
  - Arm collaboration from 2014 supported by DARPA
  - Arm Morello prototype processor, shipped 2022 supported by UKRI
  - CHERI-RISC-V due for ratification by RISC-V International mid-2025; multiple {IP, ASIC} products announced for 2025-2026
- Today's talk:
  - What is CHERI and how does it change software security?
  - Transition efforts including Arm, Google, Microsoft, and beyond …
- http://www.cheri-cpu.org/
- Watson, et al., **CHERI: Hardware-Enabled C/C++ Memory Protection at Scale**, IEEE Security and Privacy Magazine, July-August 2024.

An early experimental FPGA-based CHERI tablet prototype running the CheriBSD operating system and applications, Cambridge, 2013.

High-performance Arm Morello chip able to run a full CHERI software stack, Cambridge, 2022

What is CHERI?

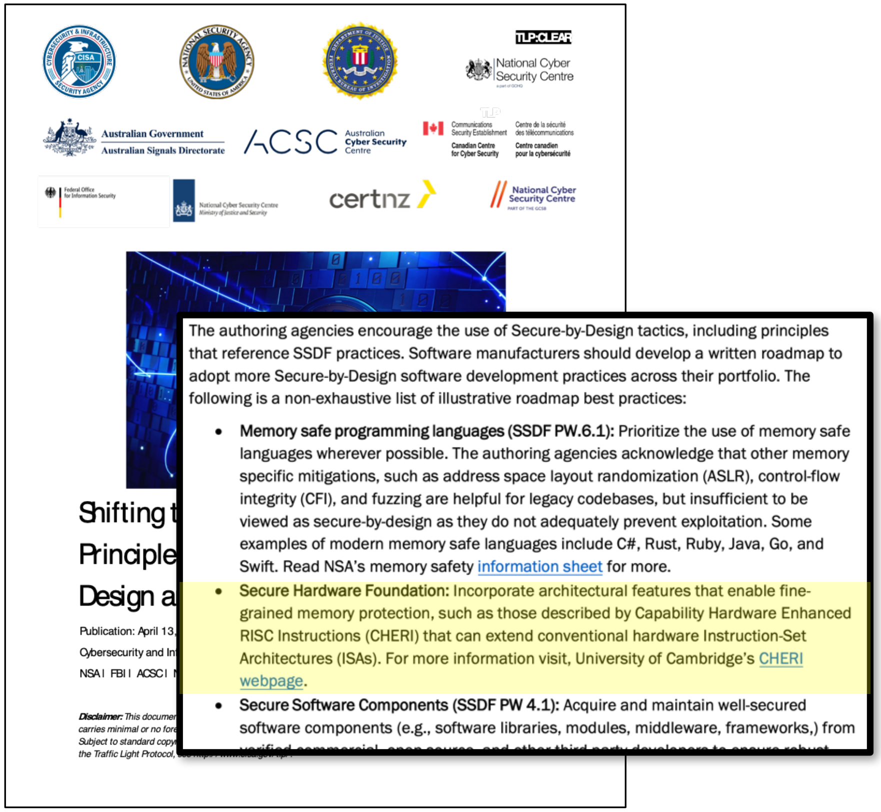


Morello chip – 7nm quad-core multi-GHz Arm processor and SoC with CHERI extensions, Arm, 2022.

- CHERI is a processor **architectural protection model**
  - Composes a **capability-system model** with hardware and software
  - Adds new security primitives to Instruction-Set Architectures (ISAs)
  - Implemented by microarchitectural extensions to the CPU and SoC
  - Enables new security behavior in software
- CHERI mitigates vulnerabilities in **C/C++ Trusted Computing Bases**
  - Hypervisors, operating systems, language runtimes, browsers, ….
  - **Fine-grained memory protection** deterministically closes many arbitrary code execution attacks, and directly impedes common exploit-chain tools
  - **Scalable compartmentalization** mitigates many vulnerability classes .. Even unknown future classes .. by extending the idea of software sandboxing
- There are now **multiple industrial implementations** (Arm, Microsoft, Codasip, …)

CHERI has featured in recent policy discussions about memory safety

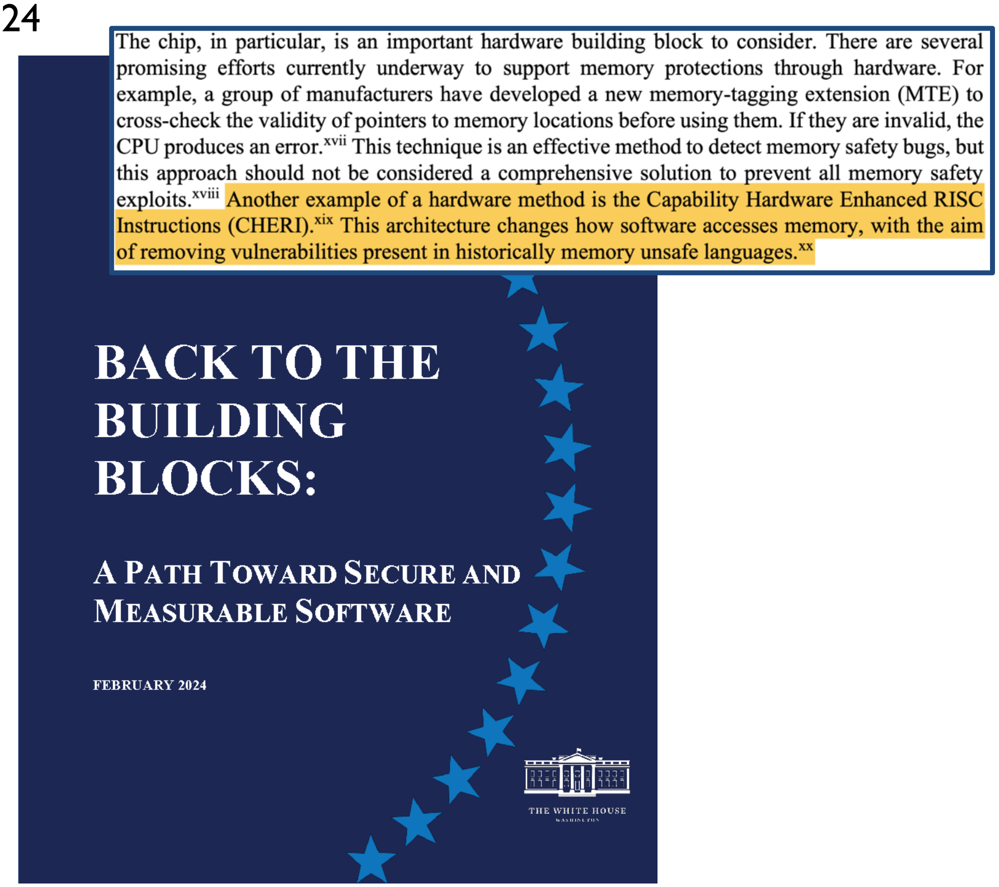
April 2023


TLP:CLEAR
National Cyber Security Centre
Australian Government
Australian Signals Directorate
ACSC Australian Cyber Security Centre
Communications Security Establishment
Centre de la sécurité des télécommunications
Canadian Centre for Cyber Security
Centre canadien pour la cybersécurité
Federal Office for Information Security
National Cyber Security Centre Ministry of Justice and Security
certnz
National Cyber Security Centre
Shifting t
Principle
Design a
Publication: April 13,
Cybersecurity and In
NSA | FBI | ACSC |

The authoring agencies encourage the use of Secure-by-Design tactics, including principles that reference SSDF practices. Software manufacturers should develop a written roadmap to adopt more Secure-by-Design software development practices across their portfolio. The following is a non-exhaustive list of illustrative roadmap best practices:

- **Memory safe programming languages (SSDF PW.6.1):** Prioritize the use of memory safe languages wherever possible. The authoring agencies acknowledge that other memory specific mitigations, such as address space layout randomization (ASLR), control-flow integrity (CFI), and fuzzing are helpful for legacy codebases, but insufficient to be viewed as secure-by-design as they do not adequately prevent exploitation. Some examples of modern memory safe languages include C#, Rust, Ruby, Java, Go, and Swift. Read NSA's memory safety information sheet for more.
- **Secure Hardware Foundation:** Incorporate architectural features that enable fine-grained memory protection, such as those described by Capability Hardware Enhanced RISC Instructions (CHERI) that can extend conventional hardware Instruction-Set Architectures (ISAs). For more information visit, University of Cambridge's CHERI webpage.
- **Secure Software Components (SSDF PW 4.1):** Acquire and maintain well-secured software components (e.g., software libraries, modules, middleware, frameworks,) from verified commercial, open source, and other third-party developers to ensure robust

February 2024



The chip, in particular, is an important hardware building block to consider. There are several promising efforts currently underway to support memory protections through hardware. For example, a group of manufacturers have developed a new memory-tagging extension (MTE) to cross-check the validity of pointers to memory locations before using them. If they are invalid, the CPU produces an error.[xvii] This technique is an effective method to detect memory safety bugs, but this approach should not be considered a comprehensive solution to prevent all memory safety exploits.[xviii] Another example of a hardware method is the Capability Hardware Enhanced RISC Instructions (CHERI).[xix] This architecture changes how software accesses memory, with the aim of removing vulnerabilities present in historically memory unsafe languages.[xx]

BACK TO THE BUILDING BLOCKS:

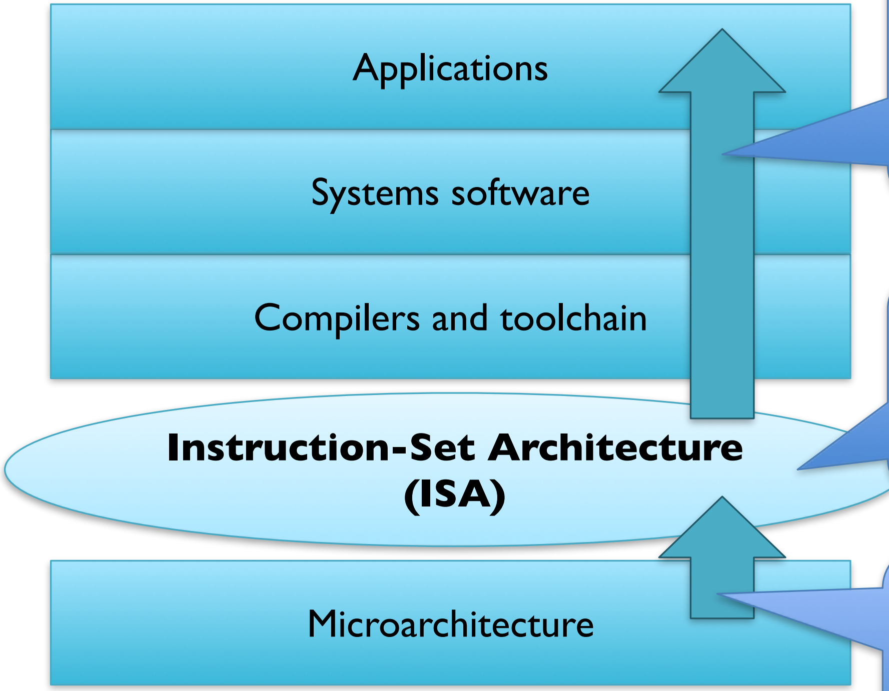
A PATH TOWARD SECURE AND MEASURABLE SOFTWARE

FEBRUARY 2024

THE WHITE HOUSE
WASHINGTON

HARDWARE-SOFTWARE CO-DESIGN FOR CHERI

Architectural primitives for software security

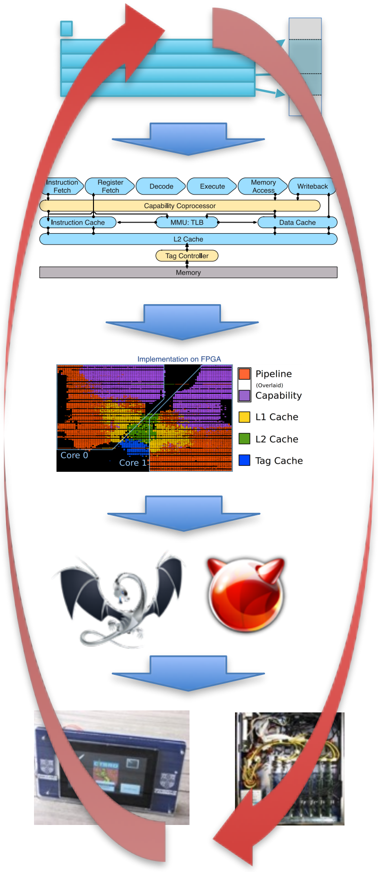


Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

CHERI capabilities are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

Hardware-software-semantics co-design



- Hardware-software-semantics co-design + concrete prototyping:
  - Portable CHERI abstract protection model
  - Concrete ISA instantiations in 32/64-bit RISC-V (+ Microsoft CHERIoT), 64-bit Armv8-a (Morello)
  - Formal ISA models, QEMU-CHERI, Morello and FPGA prototypes
  - Formal proofs that ISA security properties are met, automatic testing of simulators and implementations
  - CHERI Clang/LLVM/LLD, CheriBSD, C/C++-language applications
- Repeated iteration to improve {performance, security, compatibility, …} as evaluated through multi-million LoC studies
- Co-design includes extensive engagement with industrial partners such as Arm, Google, and Microsoft, as well as RISC-V community

CHERI ISA refinement over 14 years

Technical Report

UCAM-CL-TR-987
ISSN 1476-2986

Number 987

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

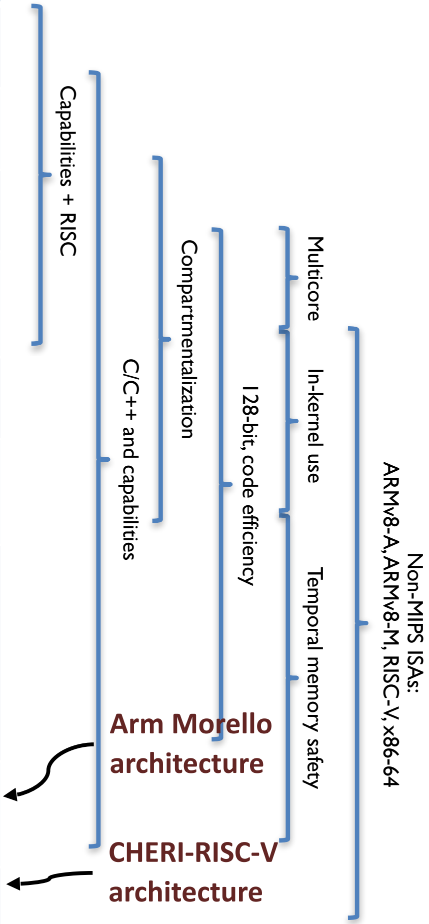
Capability Hardware
Enhanced RISC Instructions:
CHERI Instruction-Set Architecture
(Version 9)

Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff,
Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin,
Graeme Barnes, David Chisnall, Jessica Clarke, Brooks Davis,
Lee Eisen, Nathaniel Wesley Filardo, Franz A. Fuchs,
Richard Grisenthwaite, Alexandre Joannou, Ben Laurie,
A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch,
Kyndylan Nienhuis, Robert Norton, Alexander Richardson,
Peter Rugg, Peter Sewell, Stacey Son, Hongyan Xia

September 2023

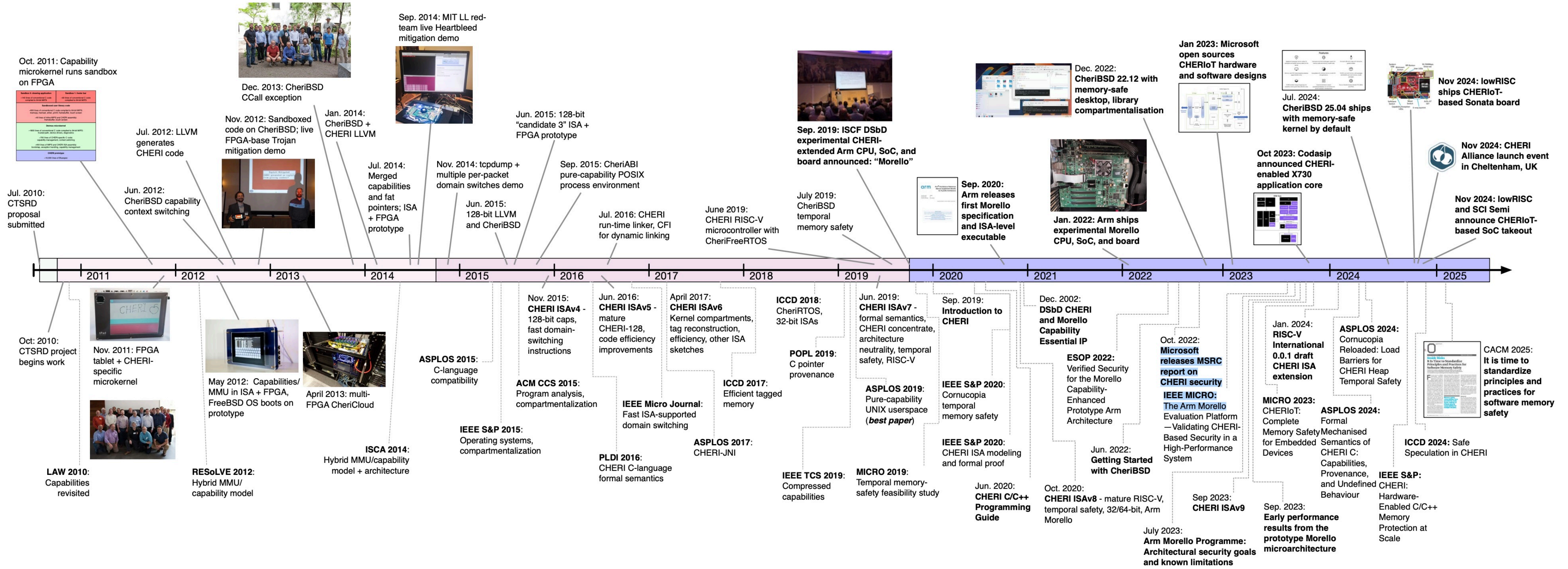
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
https://www.cl.cam.ac.uk/

Year	Version	Description
2010-2012	ISAv1	RISC capability-system model w/64-bit MIPS Capability registers, tagged memory Guarded manipulation of registers
2012	ISAv2	Extended tagging to capability registers Capability-aware exception handling Boots an MMU-based OS with CHERI support
2014	ISAv3	Fat pointers + capabilities, compiler support Instructions to optimize hybrid code Sealed capabilities, CCall/CReturn
2015	ISAv4	MMU-CHERI integration (TLB permissions) ISA support for compressed 128-bit capabilities HW-accelerated domain switching Multicore instructions: full suite of LL/SC variants
2016	ISAv5	CHERI-128 compressed capability model Improved generated code efficiency Initial in-kernel privilege limitations
2017	ISAv6	Mature kernel privilege limitations Further generated code efficiency Architectural portability: CHERI-x86, CHERI-RISC-V sketches Exception-free domain transition
2019	ISAv7	Architectural performance optimization for C++ applications Microarchitectural side-channel resistance features Architecture-neutral CHERI protection model All instruction pseudocode from a formal model CHERI Concentrate capability compression Improved C-language support, dynamic linking, sentry capabilities Elaborated CHERI-RISC-V ISA 64-bit capabilities for 32-bit architectures Accelerated tag operations for temporal memory safety
2020	ISAv8	MMU temporal memory-safety assist; e.g., capability dirty bit Optimizations for sentry capabilities CHERI-RISC-V privileged support, general maturity Further C-language semantics improvements
2023	ISAv9	CHERI-RISC-V now the reference architecture CHERI-RISC-V maturity for standardization, including tag stripping CHERI-x86 userspace sketch maturity



Watson, et al. **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 9)**, UCAM-CL-TR-987, September 2023.

CHERI research and development timeline



Jul. 2010: CTSRD proposal submitted

Oct. 2011: Capability microkernel runs sandbox on FPGA

Jun. 2012: CheriBSD capability context switching

Jul. 2012: LLVM generates CHERI code

Nov. 2012: Sandboxed code on CheriBSD; live FPGA-base Trojan mitigation demo

Dec. 2013: CheriBSD CCall exception

Jan. 2014: CheriBSD + CHERI LLVM

Jul. 2014: Merged capabilities and fat pointers; ISA + FPGA prototype

Sep. 2014: MIT LL red-team live Heartbleed mitigation demo

Nov. 2014: tcpdump + multiple per-packet domain switches demo

Jun. 2015: 128-bit LLVM and CheriBSD

Jun. 2015: 128-bit "candidate 3" ISA + FPGA prototype

Sep. 2015: CheriABI pure-capability POSIX process environment

Jul. 2016: CHERI run-time linker, CFI for dynamic linking

June 2019: CHERI RISC-V microcontroller with CheriFreeRTOS

July 2019: CheriBSD temporal memory safety

Sep. 2019: ISCF DSbD experimental CHERI-extended Arm CPU, SoC, and board announced: "Morello"

Sep. 2020: Arm releases first Morello specification and ISA-level executable

Jan. 2022: Arm ships experimental Morello CPU, SoC, and board

Dec. 2022: **CheriBSD 22.12 with memory-safe desktop, library compartmentalisation**

Jan 2023: Microsoft open sources CHERIoT hardware and software designs

Oct 2023: Codasip announced CHERI-enabled X730 application core

Jul. 2024: **CheriBSD 25.04 ships with memory-safe kernel by default**

Nov 2024: lowRISC ships CHERIoT-based Sonata board

Nov 2024: CHERI Alliance launch event in Cheltenham, UK

Nov 2024: lowRISC and SCI Semi announce CHERIoT-based SoC takeout

Oct: 2010: CTSRD project begins work

LAW 2010: Capabilities revisited

Nov. 2011: FPGA tablet + CHERI-specific microkernel

May 2012: Capabilities/ MMU in ISA + FPGA, FreeBSD OS boots on prototype

RESoLVE 2012: Hybrid MMU/ capability model

April 2013: multi-FPGA CheriCloud

ISCA 2014: Hybrid MMU/capability model + architecture

ASPLOS 2015: C-language compatibility

IEEE S&P 2015: Operating systems, compartmentalization

ACM CCS 2015: Program analysis, compartmentalization

Nov. 2015: **CHERI ISAv4 -** 128-bit caps, fast domain-switching instructions

Jun. 2016: **CHERI ISAv5 -** mature CHERI-128, code efficiency improvements

PLDI 2016: CHERI C-language formal semantics

IEEE Micro Journal: Fast ISA-supported domain switching

April 2017: **CHERI ISAv6** Kernel compartments, tag reconstruction, efficiency, other ISA sketches

ASPLOS 2017: CHERI-JNI

ICCD 2017: Efficient tagged memory

ICCD 2018: CheriRTOS, 32-bit ISAs

POPL 2019: C pointer provenance

IEEE TCS 2019: Compressed capabilities

Jun. 2019: **CHERI ISAv7 -** formal semantics, CHERI concentrate, architecture neutrality, temporal safety, RISC-V

ASPLOS 2019: Pure-capability UNIX userspace (***best paper***)

MICRO 2019: Temporal memory-safety feasibility study

Sep. 2019: **Introduction to CHERI**

IEEE S&P 2020: Cornucopia temporal memory safety

IEEE S&P 2020: CHERI ISA modeling and formal proof

Jun. 2020: **CHERI C/C++ Programming Guide**

Dec. 2002: **DSbD CHERI and Morello Capability Essential IP**

Oct. 2020: **CHERI ISAv8** - mature RISC-V, temporal safety, 32/64-bit, Arm Morello

ESOP 2022: Verified Security for the Morello Capability-Enhanced Prototype Arm Architecture

Jun. 2022: **Getting Started with CheriBSD**

Oct. 2022: **Microsoft releases MSRC report on CHERI security**

IEEE MICRO: The Arm Morello Evaluation Platform—Validating CHERI-Based Security in a High-Performance System

July 2023: **Arm Morello Programme: Architectural security goals and known limitations**

Sep 2023: **CHERI ISAv9**

Sep. 2023: **Early performance results from the prototype Morello microarchitecture**

MICRO 2023: CHERIoT: Complete Memory Safety for Embedded Devices

Jan. 2024: **RISC-V International 0.0.1 draft CHERI ISA extension**

ASPLOS 2024: Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety

ASPLOS 2024: Formal Mechanised Semantics of CHERI C: Capabilities, Provenance, and Undefined Behaviour

IEEE S&P: CHERI: Hardware-Enabled C/C++ Memory Protection at Scale

ICCD 2024: Safe Speculation in CHERI

CACM 2025: **It is time to standardize principles and practices for software memory safety**

Years 1-2: Research platform, prototype architecture

Years 2-4: Hybrid C/OS, compartment model

Years 4-7: Efficiency, CheriABI/C/C++/linker, Armv8-A

Years 8-12: RISC-V, temporal safety, proofs, Arm Morello, Microsoft CHERIoT Ibex

Years 13-14: CHERI software ecosystem, library & kernel compartmentalization, commercial products, RISC-V standardization

CHERI

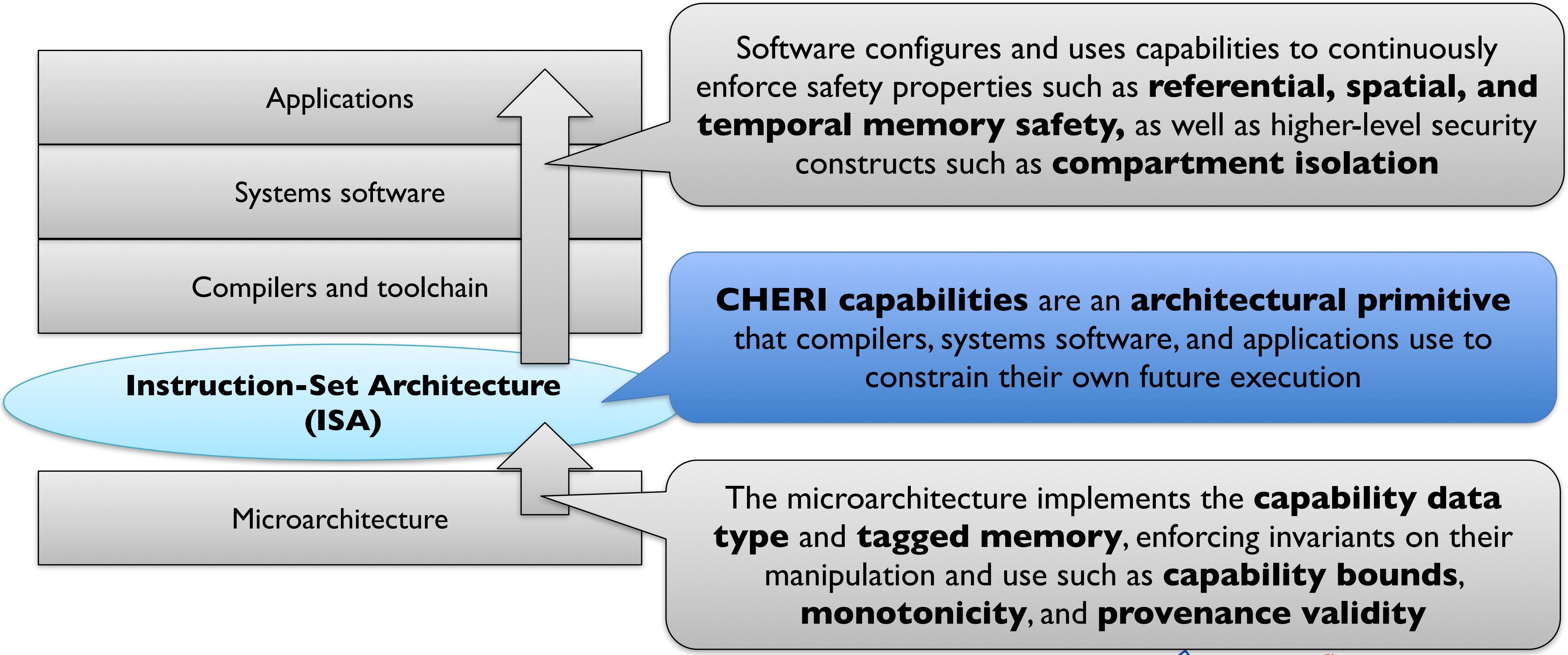
SRI

CAPABILITIES LIMITED

UNIVERSITY OF CAMBRIDGE

CHERI PROTECTION MODEL AND ARCHITECTURE

Architectural primitives for software security

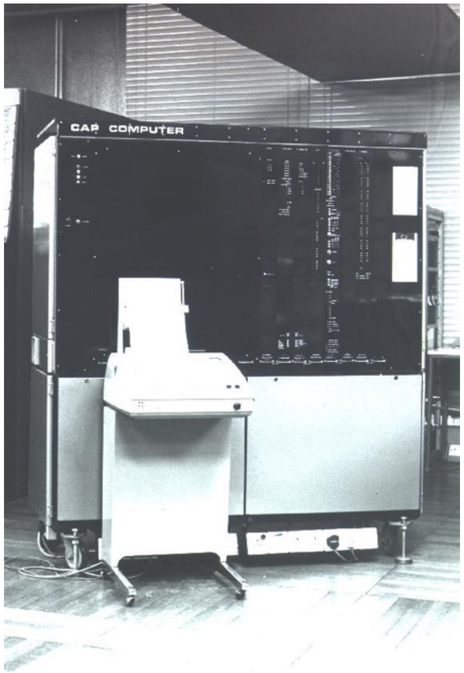


Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

CHERI capabilities are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

Capability systems



The CAP computer project ran from 1970-1977 at the University of Cambridge, led by R. Needham, M. Wilkes, and D. Wheeler.

- The capability system is a **design pattern** for how CPUs, languages, OSes, … can control access to resources
  - **Capabilities** are communicable, unforgeable tokens of authority
  - In **capability-based systems,** resources are reachable **only** via capabilities
- Capability systems limit the **scope and spread of damage** from accidental or intentional software misbehavior
- They do this by making it **natural and efficient** to implement, in software, two security design principles:
  - The **principle of least privilege** dictates that software should run with the minimum privileges to perform its tasks
  - The **principle of intentional use** dictates that when software holds multiple privileges, it must explicitly select which to exercise
- **These two principles are the heart of the CHERI design**

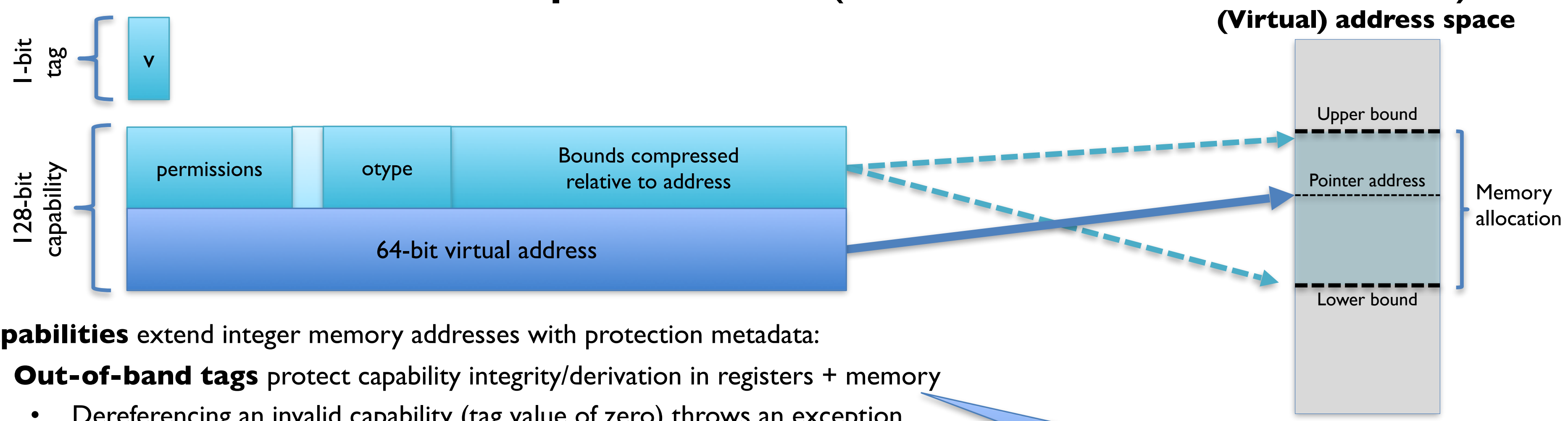
CHERI design goals and approach

- **De-conflate memory virtualization and protection**
  - Memory Management Units (MMUs) protect by **location (address)**
  - CHERI protects existing **references (pointers)** to code, data, objects
  - Reusing **existing pointer indirection** avoids adding new architectural table lookups
- **Architectural mechanism** that enforces **software policies**
  - **Language-based properties** – e.g., referential, spatial, and temporal integrity (C/C++ compiler, linkers, OS model, runtime, …)
  - **New software abstractions** – e.g., software compartmentalization (confined objects for in-address-space isolation, …)

CHERI is a portable model

- CHERI is an **abstract architectural protection model**; like VM, it is:
  - Integrated into the **Instruction-Set Architecture (ISA)**
  - **ISA neutral** – (thus far) applied to both 32-bit and 64-bit architectures; e.g., MIPS, RISC-V, Arm-A, Arm-M, and x86_64
  - Accommodates **diverse microarchitectures** (e.g., pipelined, OoO,, …)
  - Has a **portable software model** added to multiple compilers, OSes, etc.
- Various CHERI-enabled software projects aim to start upstreaming support as first production microarchitectures ship over the next 18 months
  - Most changes are **CHERI-specific** rather than "CHERI-RISC-V-specific" or "Morello-specific"; e.g., CHERI LLVM, CHERI GDB, CheriBSD, CHERI Linux, CHERI-seL4, …
  - Portions of compiler back ends, debugger targets, OS assembly, etc., are ISA-specific
- While the initial focus for CHERI was on the C/C++ corpus, there is substantial work on using CHERI for other languages and language runtimes; e.g., Rust, OpenJDK, V8

CHERI 128-bit capabilities (64-bit, MMU-enabled)

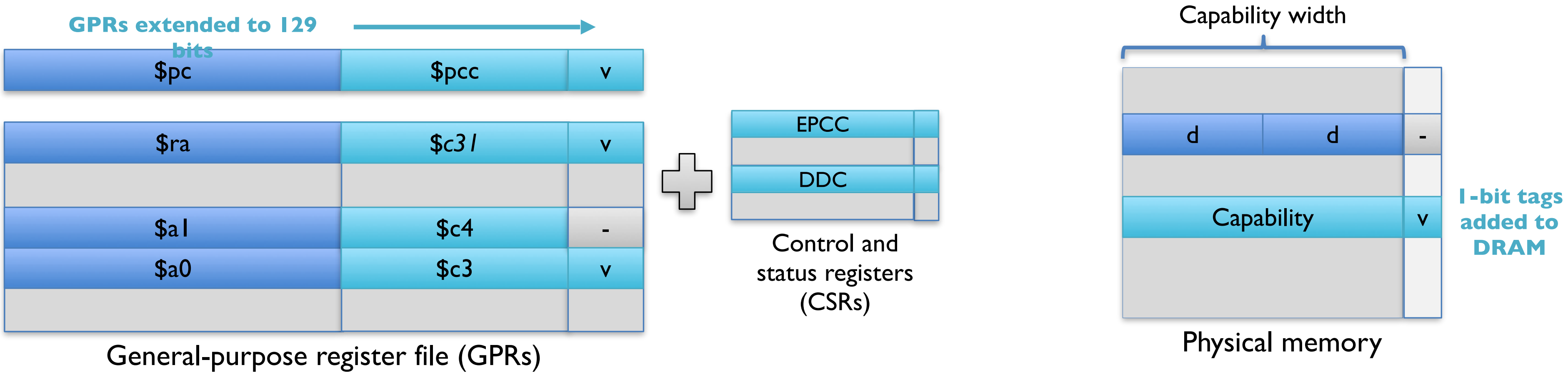


Capabilities extend integer memory addresses with protection metadata:

- **Out-of-band tags** protect capability integrity/derivation in registers + memory
  - Dereferencing an invalid capability (tag value of zero) throws an exception
  - Overwriting a capability in memory clears its validity tag
- **Bounds** and **permissions** authorize access to memory
  - Dereferencing a capability outside of its bounds, permissions, etc., throws an exception
- **Sealing** implements non-bypassable encapsulation for data and control flow
- **Guarded manipulation** controls how capability values themselves may be manipulated
  - E.g., enforcing **provenance validity** and **monotonicity**

CHERI's tag enables **deterministic**, **secrets-free** protection

Capability-extended register file + tagged memory



GPRs extended to 129 bits

$pc	$pcc	v

$ra	$c31	v
$a1	$c4	-
$a0	$c3	v

General-purpose register file (GPRs)

EPCC	
DDC	

Control and status registers (CSRs)

Capability width

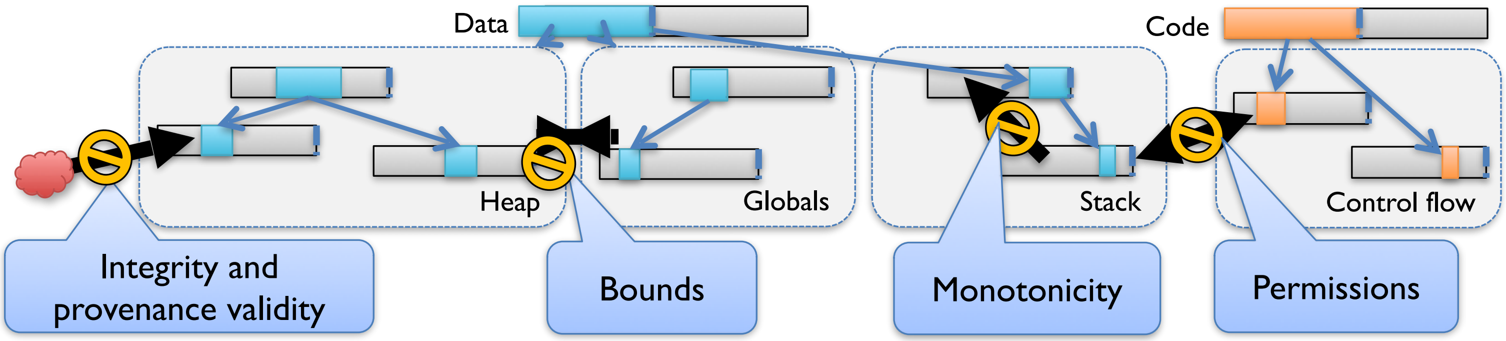
d	d	-
Capability		v

1-bit tags added to DRAM

Physical memory

- **64-bit general-purpose registers (GPRs)** extended with **64 bits of metadata** and a **1-bit validity tag**
- **Program counter (PC)** is extended to be the **program-counter capability ($PCC)**
- **Tagged memory** protects capability-sized and -aligned words in DRAM by adding a **1-bit validity tag**
- **New instructions** are used to explicitly load, store, inspect, and manipulate capability values
- **Existing encodings** are reused for capability-relative dereferences when in a suitable mode
- **Default data capability ($DDC)** constrains legacy integer-relative ISA load and store instructions
- **System mechanisms** are extended (e.g., capability-instruction enable control register, new PTE permissions, new exception codes, exception stack pointers + vectors become capabilities, etc.)

Capability semantics applied to pointers

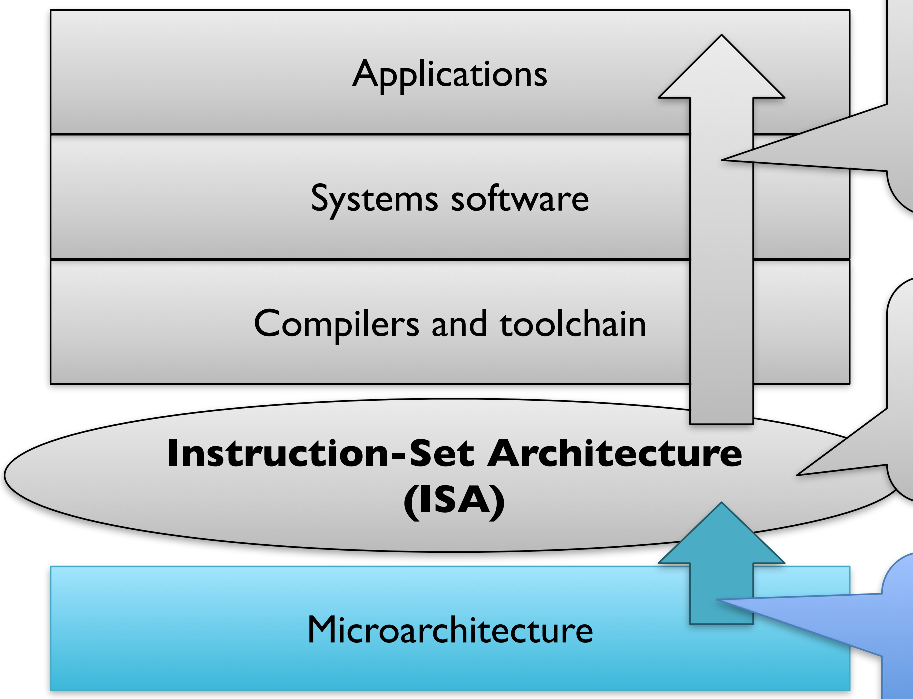


- **Tags** protect the **integrity** and **provenance validity** of pointers by:
  - Constraining manipulation, detecting corruption, and preventing injection (e.g., via the network)
  - Enabling accurate and deterministic detection, efficient tracking and revocation (i.e., temporal safety)
- **Bounds** prevent pointers from being used to access the wrong object (i.e., spatial safety)
- **Monotonicity** prevents pointer privilege escalation (e.g., broadening bounds)
- **Permissions** limit unintended use of pointers (e.g., W^X for pointers)
- **Sealing** prevents dereferencing, and enables non-monotonic domain transition

→ **Deterministic, secrets-free memory protection** and **scalable software compartmentalization**

CHERI MICROARCHITECTURE AND PROTOTYPES

Architectural primitives for software security



Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

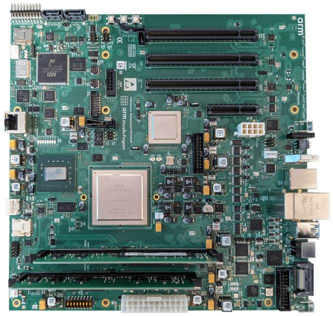
CHERI capabilities are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

CHERI demonstrated at a range of scales

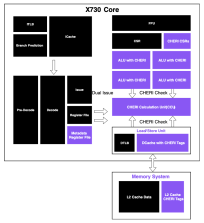
Innovate UK

Mobile devices, data centers



Arm Morello application core + SoC, based on Neoverse N1
64-bit Arm-A baseline ISA
Multicore, MMU-enabled, out-of-order core 2.5GHz
CHERI-adapted FreeBSD, Linux, seL4, VxWorks OSes

Automotive, embedded, high-end IoT



Codasip X730 application core, based on A730
64-bit RISC-V baseline ISA
Dual-issue, pipelined, with MMU
CHERI-adapted FreeBSD, Linux, seL4 OSes

IoT, roots of trust

Microsoft CHERIoT Ibex microcontroller
32-bit RISC-V baseline ISA
3-stage pipeline, no MMU, 200-300MHz
CHERIoT RTOS embedded OS

Secure Compute Institute

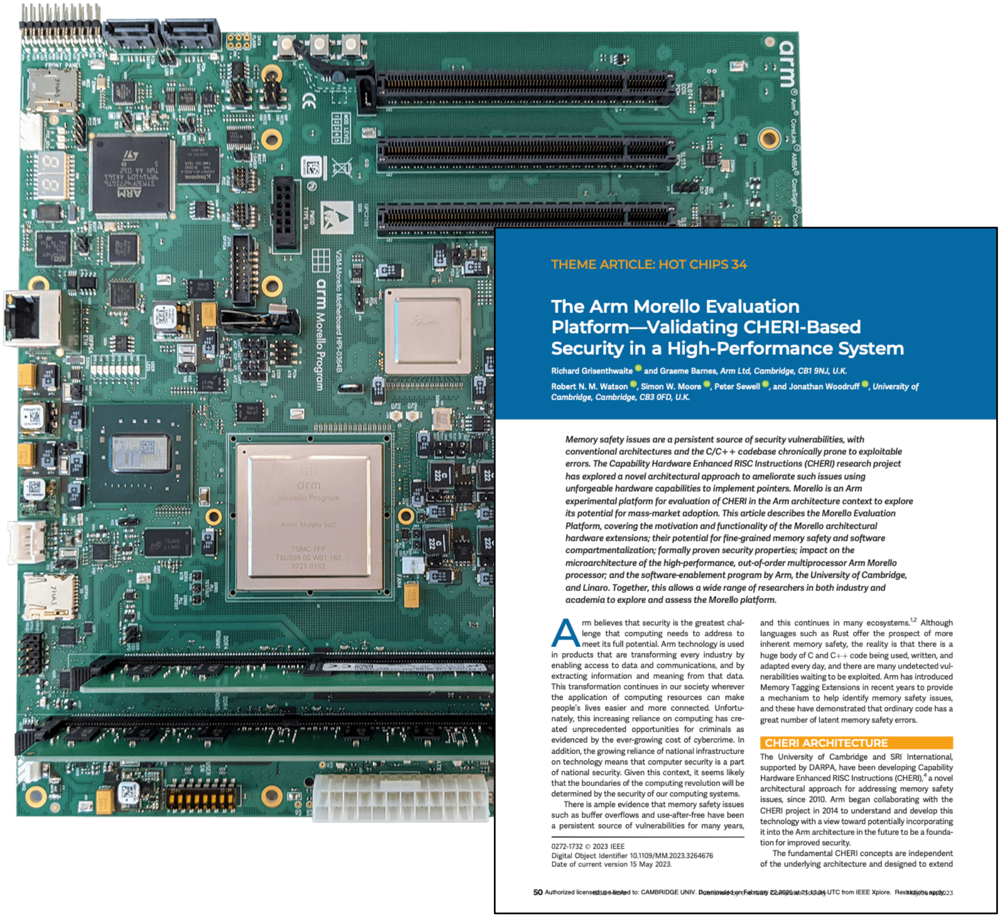
CHERI Supported by InnovateUK as part of Digital Security by Design (DSbD)

CAPABILITIES LIMITED

Arm Morello (2022)

UKRI

Innovate UK



THEME ARTICLE: HOT CHIPS 34

The Arm Morello Evaluation Platform—Validating CHERI-Based Security in a High-Performance System

Richard Grisenthwaite and Graeme Barnes, *Arm Ltd, Cambridge, CB1 9NJ, U.K.*

Robert N. M. Watson, Simon W. Moore, Peter Sewell, and Jonathan Woodruff, *University of Cambridge, Cambridge, CB3 0FD, U.K.*

Memory safety issues are a persistent source of security vulnerabilities, with conventional architectures and the C/C++ codebase chronically prone to exploitable errors. The Capability Hardware Enhanced RISC Instructions (CHERI) research project has explored a novel architectural approach to ameliorate such issues using unforgeable hardware capabilities to implement pointers. Morello is an Arm experimental platform for evaluation of CHERI in the Arm architecture context to explore its potential for mass-market adoption. This article describes the Morello Evaluation Platform, covering the motivation and functionality of the Morello architectural hardware extensions; their potential for fine-grained memory safety and software compartmentalization; formally proven security properties; impact on the microarchitecture of the high-performance, out-of-order multiprocessor Arm Morello processor; and the software-enablement program by Arm, the University of Cambridge, and Linaro. Together, this allows a wide range of researchers in both industry and academia to explore and assess the Morello platform.

Arm believes that security is the greatest challenge that computing needs to address to meet its full potential. Arm technology is used in products that are transforming every industry by enabling access to data and communications, and by extracting information and meaning from that data. This transformation continues in our society wherever the application of computing resources can make people's lives easier and more connected. Unfortunately, this increasing reliance on computing has created unprecedented opportunities for criminals as evidenced by the ever-growing cost of cybercrime. In addition, the growing reliance of national infrastructure on technology means that computer security is a part of national security. Given this context, it seems likely that the boundaries of the computing revolution will be determined by the security of our computing systems.

There is ample evidence that memory safety issues such as buffer overflows and use-after-free have been a persistent source of vulnerabilities for many years, and this continues in many ecosystems.[1,2] Although languages such as Rust offer the prospect of more inherent memory safety, the reality is that there is a huge body of C and C++ code being used, written, and adapted every day, and there are many undetected vulnerabilities waiting to be exploited. Arm has introduced Memory Tagging Extensions in recent years to provide a mechanism to help identify memory safety issues, and these have demonstrated that ordinary code has a great number of latent memory safety errors.

CHERI ARCHITECTURE

The University of Cambridge and SRI International, supported by DARPA, have been developing Capability Hardware Enhanced RISC Instructions (CHERI),[4] a novel architectural approach for addressing memory safety issues, since 2010. Arm began collaborating with the CHERI project in 2014 to understand and develop this technology with a view toward potentially incorporating it into the Arm architecture in the future to be a foundation for improved security.

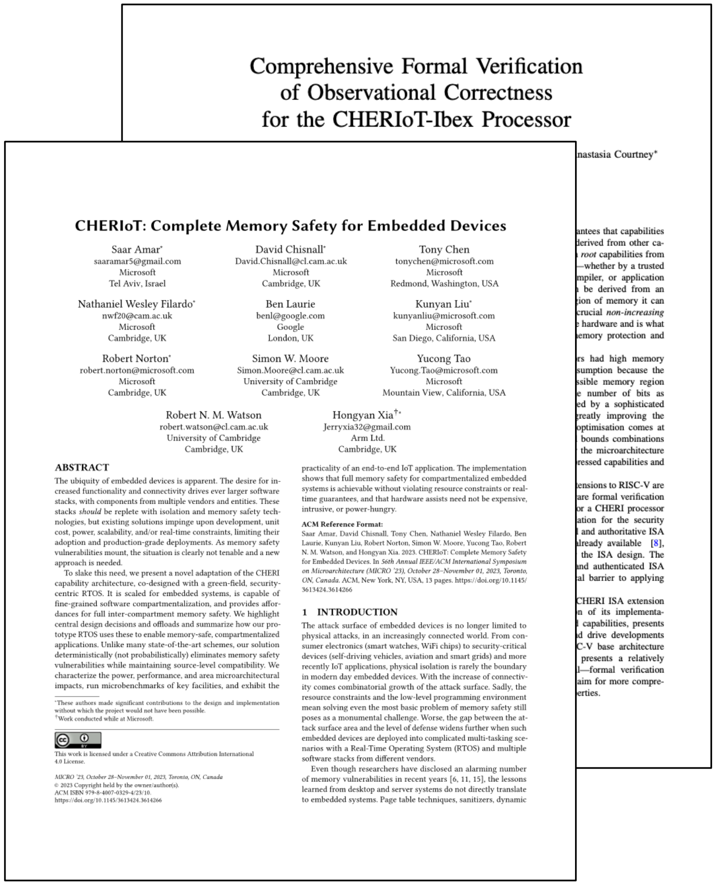
The fundamental CHERI concepts are independent of the underlying architecture and designed to extend

0272-1732 © 2023 IEEE
Digital Object Identifier 10.1109/MM.2023.3264676
Date of current version 15 May 2023.

IEEE Micro Journal 2023

- $225M government, academia, and industrial research program led by UK Research and Innovation (UKRI)
  - Announced partners: Arm, Google, Microsoft
  - 15+ UK universities with research grants
  - 70+ funded business incubation projects
- Baseline for design: Neoverse N1 core
  - 2.5GHz quad-core, superscalar
  - Implements CHERI extensions
  - Runs full CHERI-enabled software stacks
  - Definitely a prototype, but a very powerful one!
- Roughly a thousand chips manufactured for use by research + development labs

Microsoft CHERIoT core (2023)



Comprehensive Formal Verification
of Observational Correctness
for the CHERIoT-Ibex Processor

nastasia Courtney*

antees that capabilities
derived from other ca-
root capabilities from
—whether by a trusted
mpiler, or application
be derived from an
ion of memory it can
crucial *non-increasing*
hardware and is what
emory protection and

rs had high memory
sumption because the
ssible memory region
e number of bits as
ed by a sophisticated
reatly improving the
optimisation comes at
bounds combinations
the microarchitecture
ressed capabilities and

ensions to RISC-V are
are formal verification
or a CHERI processor
ation for the security
and authoritative ISA
lready available [8],
the ISA design. The
nd authenticated ISA
al barrier to applying

CHERI ISA extension
n of its implementa-
capabilities, presents
d drive developments
C-V base architecture
presents a relatively
l—formal verification
aim for more compre-
erties.

CHERIoT: Complete Memory Safety for Embedded Devices

Saar Amar[*]
saaramar5@gmail.com
Microsoft
Tel Aviv, Israel

David Chisnall[*]
David.Chisnall@cl.cam.ac.uk
Microsoft
Cambridge, UK

Tony Chen
tonychen@microsoft.com
Microsoft
Redmond, Washington, USA

Nathaniel Wesley Filardo[*]
nwf20@cam.ac.uk
Microsoft
Cambridge, UK

Ben Laurie
benl@google.com
Google
London, UK

Kunyan Liu[*]
kunyanliu@microsoft.com
Microsoft
San Diego, California, USA

Robert Norton[*]
robert.norton@microsoft.com
Microsoft
Cambridge, UK

Simon W. Moore
Simon.Moore@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Yucong Tao
Yucong.Tao@microsoft.com
Microsoft
Mountain View, California, USA

Robert N. M. Watson
robert.watson@cl.cam.ac.uk
University of Cambridge
Cambridge, UK

Hongyan Xia[†*]
Jerryxia32@gmail.com
Arm Ltd.
Cambridge, UK

ABSTRACT

The ubiquity of embedded devices is apparent. The desire for increased functionality and connectivity drives ever larger software stacks, with components from multiple vendors and entities. These stacks *should* be replete with isolation and memory safety technologies, but existing solutions impinge upon development, unit cost, power, scalability, and/or real-time constraints, limiting their adoption and production-grade deployments. As memory safety vulnerabilities mount, the situation is clearly not tenable and a new approach is needed.

To slake this need, we present a novel adaptation of the CHERI capability architecture, co-designed with a green-field, security-centric RTOS. It is scaled for embedded systems, is capable of fine-grained software compartmentalization, and provides affordances for full inter-compartment memory safety. We highlight central design decisions and offloads and summarize how our prototype RTOS uses these to enable memory-safe, compartmentalized applications. Unlike many state-of-the-art schemes, our solution deterministically (not probabilistically) eliminates memory safety vulnerabilities while maintaining source-level compatibility. We characterize the power, performance, and area microarchitectural impacts, run microbenchmarks of key facilities, and exhibit the practicality of an end-to-end IoT application. The implementation shows that full memory safety for compartmentalized embedded systems is achievable without violating resource constraints or real-time guarantees, and that hardware assists need not be expensive, intrusive, or power-hungry.

ACM Reference Format:
Saar Amar, David Chisnall, Tony Chen, Nathaniel Wesley Filardo, Ben Laurie, Kunyan Liu, Robert Norton, Simon W. Moore, Yucong Tao, Robert N. M. Watson, and Hongyan Xia. 2023. CHERIoT: Complete Memory Safety for Embedded Devices. In *56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23), October 28–November 01, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3613424.3614266

1 INTRODUCTION

The attack surface of embedded devices is no longer limited to physical attacks, in an increasingly connected world. From consumer electronics (smart watches, WiFi chips) to security-critical devices (self-driving vehicles, aviation and smart grids) and more recently IoT applications, physical isolation is rarely the boundary in modern day embedded devices. With the increase of connectivity comes combinatorial growth of the attack surface. Sadly, the resource constraints and the low-level programming environment mean solving even the most basic problem of memory safety still poses as a monumental challenge. Worse, the gap between the attack surface area and the level of defense widens further when such embedded devices are deployed into complicated multi-tasking scenarios with a Real-Time Operating System (RTOS) and multiple software stacks from different vendors.

Even though researchers have disclosed an alarming number of memory vulnerabilities in recent years [6, 11, 15], the lessons learned from desktop and server systems do not directly translate to embedded systems. Page table techniques, sanitizers, dynamic

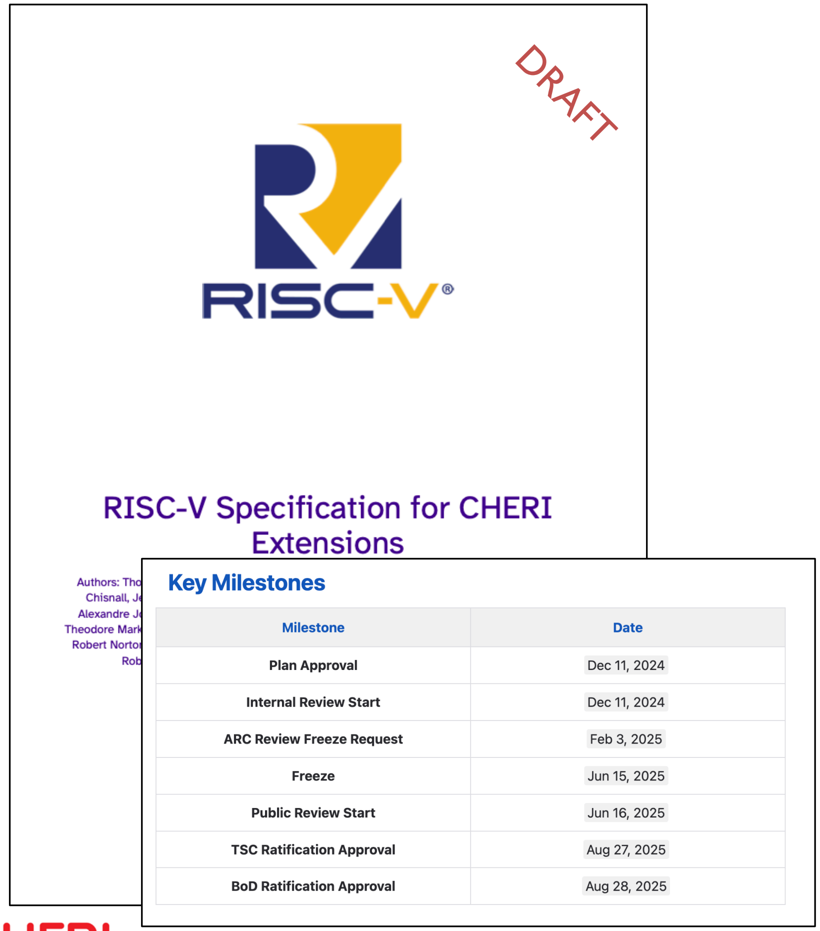
[*] These authors made significant contributions to the design and implementation without which the project would not have been possible.
[†] Work conducted while at Microsoft.

MICRO '23, October 28–November 01, 2023, Toronto, ON, Canada
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0329-4/23/10.
https://doi.org/10.1145/3613424.3614266

IEEE MICRO 2023

- CHERI-extended Ibex microcontroller
  - Microcontroller used in OpenTitan, etc.
  - CHERI-RISC-V tuned for microcontrollers
  - Clean-slate memory-safe, compartmentalized embedded OS
  - CHERI extensions for revocation in SRAM
  - Open sourced in February 2023
- Development a collaboration across Microsoft Research, MSRC, Azure Silicon, and Azure Edge + Platform
- Formal verification that microarchitecture implements specified ISA completed by Oxford University and University of Cambridge in 2025
- Now shipping in the lowRISC Sonata FPGA board, and being taped out for lowRISC/SCI Semiconductor ICENI board

CHERI-RISC-V standardization



RISC-V Specification for CHERI Extensions

Authors: Tho
Chisnall, Je
Alexandre Jo
Theodore Mark
Robert Norton
Rob

Key Milestones

Milestone	Date
Plan Approval	Dec 11, 2024
Internal Review Start	Dec 11, 2024
ARC Review Freeze Request	Feb 3, 2025
Freeze	Jun 15, 2025
Public Review Start	Jun 16, 2025
TSC Ratification Approval	Aug 27, 2025
BoD Ratification Approval	Aug 28, 2025

- RISC-V International CHERI Special Interest Group (SIG) and Technical Group (TG)
  - Co-chairs Alex Richardson (GChips), Simon Moore (Cambridge)
  - Strong industrial engagement from multiple companies building products
  - Incorporates both SRI/Cambridge baseline CHERI for application cores, and also Microsoft's CHERIoT extensions for microcontrollers
- Full draft specification now in review
- Ratification planned for August 2025
- Updated versions of CHERI LLVM, QEMU, …
- Multiple processor designs targeting spec.

https://riscv.github.io/riscv-cheri/

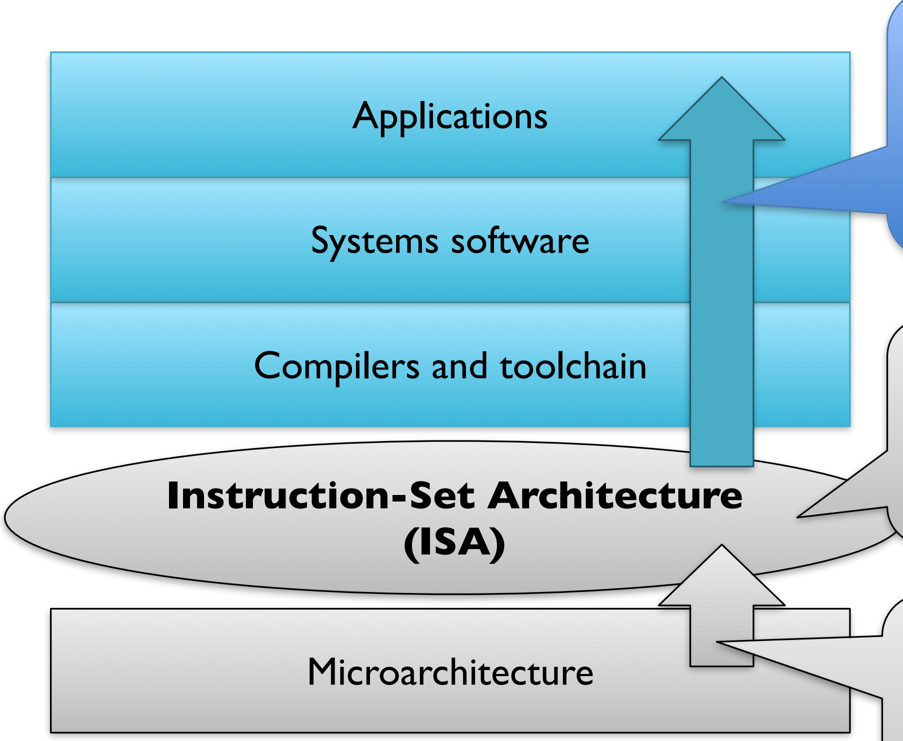
Current and arriving CHERI-RISC-V products

- **Microsoft** open-source CHERIoT-Ibex 32-bit microcontroller core IP
- **Google** CHERI-enabled open-source ML accelerator prototype
- **lowRISC** CHERIoT-based Sonata FPGA board, ICENI FPGA board, ICENI SoC
- **SCI Semiconductor** CHERIoT-based ICENI SoC
- **Secqai** CHERI Ibex post-quantum crypto communications SoC
- **Codasip** proprietary 64-bit, MMU-enabled application core IP
- **CapLtd** open-source SystemVerilog CHERI CVA6 application core IP reference design
- **University of Cambridge** open-source research BSV cores: Piccolo, Flute, Toooba

Multiple test chips for CHERI-RISC-V silicon targeting 2025/2026

HOW SOFTWARE WORKS ON CHERI

Architectural primitives for software security



Applications

Systems software

Compilers and toolchain

Instruction-Set Architecture (ISA)

Microarchitecture

Software configures and uses capabilities to continuously enforce safety properties such as **referential, spatial, and temporal memory safety,** as well as higher-level security constructs such as **compartment isolation**

CHERI capabilities are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

The microarchitecture implements the **capability data type** and **tagged memory**, enforcing invariants on their manipulation and use such as **capability bounds**, **monotonicity**, and **provenance validity**

Two key applications of the CHERI primitives

1. **Efficient, fine-grained memory protection for C/C++**
   - Strong source-level compatibility, but requires recompilation
   - Deterministic and secret-free referential, spatial, and temporal memory safety
   - Retrospective studies estimate ⅔ of memory-safety vulnerabilities mitigated
   - Generally modest overhead (0%-5%, some pointer-dense workloads higher)
2. **Scalable software compartmentalization**
   - Multiple software operational models from objects to processes
   - Increases exploit chain length: Attackers must find and exploit more vulnerabilities
   - Orders-of-magnitude performance improvement over MMU-based techniques (<90% reduction in IPC overhead in early FPGA-based benchmarks)

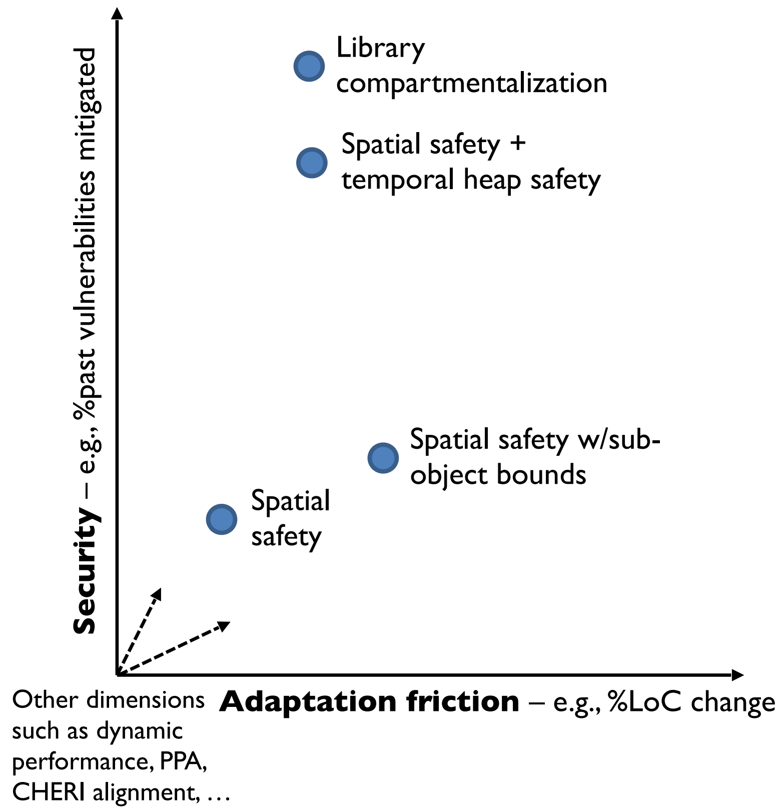
CHERI C/C++ MEMORY PROTECTION

What do we mean by C/C++ memory safety?

- Complex question, as while **memory unsafety** is clearly present, neither language defines what **memory safety** could mean
- Our thoughts from over a decade working on CHERI:
  - **Memory safety** for C/C++ is (pragmatically) anything that would have defended you from memory-safety vulnerabilities
  - **Vulnerability mitigation** deterministically coerces bugs that are currently vulnerabilities back into bugs – i.e., you would no longer urgently patch them
  - **Exploit mitigation** interferes with attack techniques that exploit memory unsafety
  - **Deterministic mitigation** means that defenses always work regardless of information leakage, attempts to brute force, and so on
- Our ambition for CHERI C/C++ memory safety is to **mitigate the vast majority (>70%) of memory-safety vulnerabilities with full determinism**
- Actual mitigation substantially exceeds this rate due to capabilities throughout the language runtime for exploit mitigation, but our field lacks methodology to evaluate this

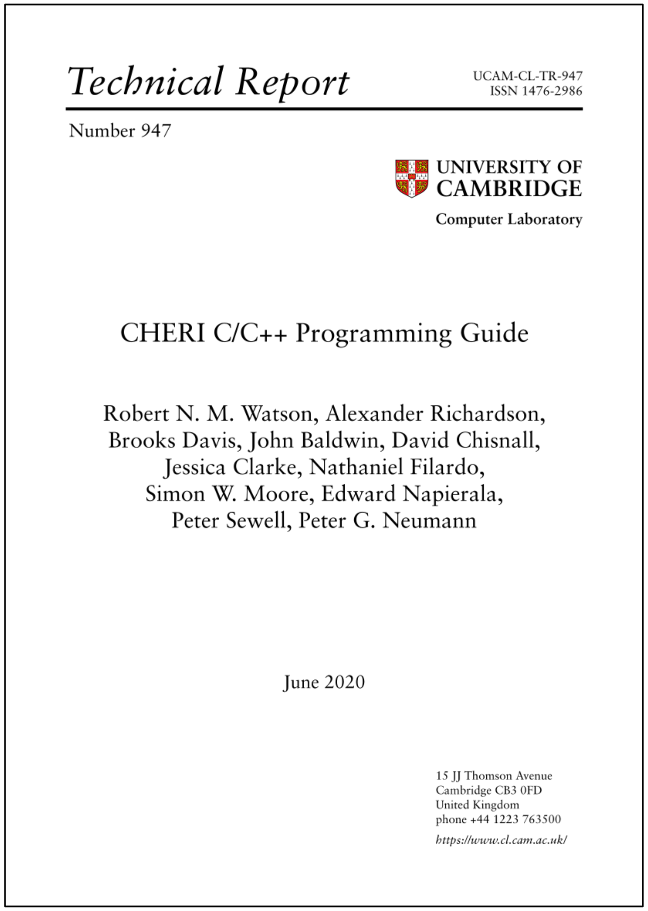
Useful definitions for CHERI C/C++ defenses, but also in comparing to other memory-safety techniques

A space of C memory-protection models


Library compartmentalization
Spatial safety + temporal heap safety
Spatial safety w/sub-object bounds
Spatial safety
Security – e.g., %past vulnerabilities mitigated
Adaptation friction – e.g., %LoC change
Other dimensions such as dynamic performance, PPA, CHERI alignment, …

- C does not define a memory-protection model
  - We have therefore had to (organically) grow one
- Optimization goals have been:
  - Works well with CHERI (changing CHERI allowed, subject to PPA)
  - %LoC source-code modification rates
  - ABI / code-generation / optimization model alignment with status quo
  - Dynamic performance overhead (e.g., cycles)
  - Vulnerability mitigation (ideally deterministic)
- There is a rich space of potential memory-protection models
  - Points combine (or not) different protection options
  - E.g., Sub-object bounds, heap/stack temporal safety, …
  - Today's trade-off point hits around 70% of memory-safety vulnerabilities
  - Compartmentalization shifts adversary model to arbitrary code execution

Memory-safe CHERI C/C++



Technical Report

UCAM-CL-TR-947
ISSN 1476-2986

Number 947

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

CHERI C/C++ Programming Guide

Robert N. M. Watson, Alexander Richardson,
Brooks Davis, John Baldwin, David Chisnall,
Jessica Clarke, Nathaniel Filardo,
Simon W. Moore, Edward Napierala,
Peter Sewell, Peter G. Neumann

June 2020

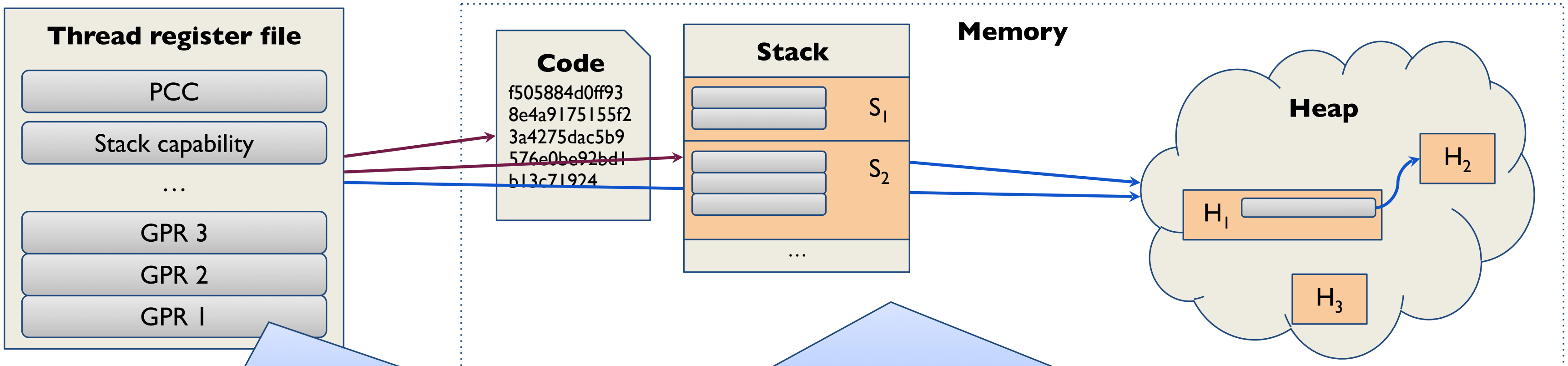
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
https://www.cl.cam.ac.uk/

- Capabilities used to implement all pointers

  Implied – Control-flow pointers, stack pointers, GOTs, PLTs, …

  Explicit – All C/C++-level pointers and references
- Strong referential, spatial, and heap temporal safety
- Minor changes to C/C++ semantics; e.g.,
  - All pointers must have well defined single provenance
  - Increased pointer size and alignment
  - Care required with integer-pointer casts and types
  - Memory-copy implementations may need to preserve tags
- Watson, et al. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020

Implementing C/C++ memory safety with CHERI (1/2)

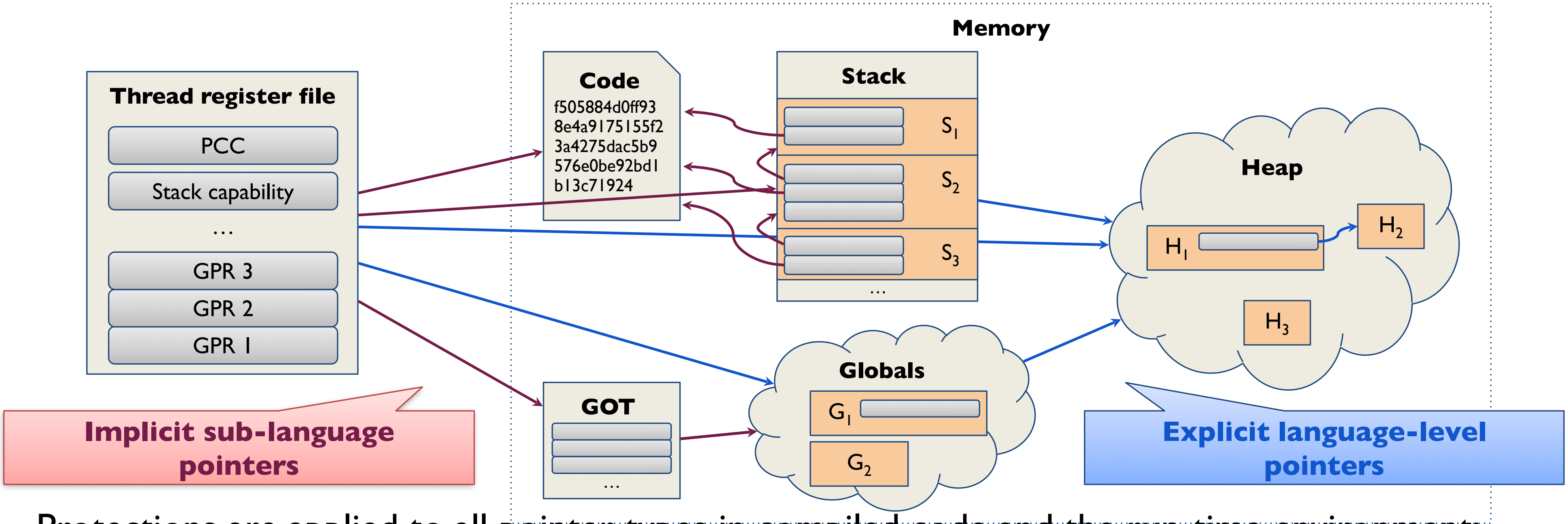


General-purposes registers and the **program counter (PC)** are extended to capability width and have tags.

Tagged memory preserves capability tag values flowing into and out of registers. Tags are **implicitly cleared** if non-capability stores overwrite tagged values.

- Whereas conventional C/C++ use **integer pointers**, CHERI C/C++ uses **capability pointers**
  - C/C++ data types are laid out for wider **capability-width, capability-aligned pointers**
  - Code generation is **pointer-aware** and uses **explicit load/store-capability instructions**
- Software TCBs restrict capability bounds and permissions as code runs
- Hardware continuously enforces capability protections on all pointer manipulation and use

Implementing C/C++ memory safety with CHERI (2/2)



Implicit sub-language pointers

Explicit language-level pointers

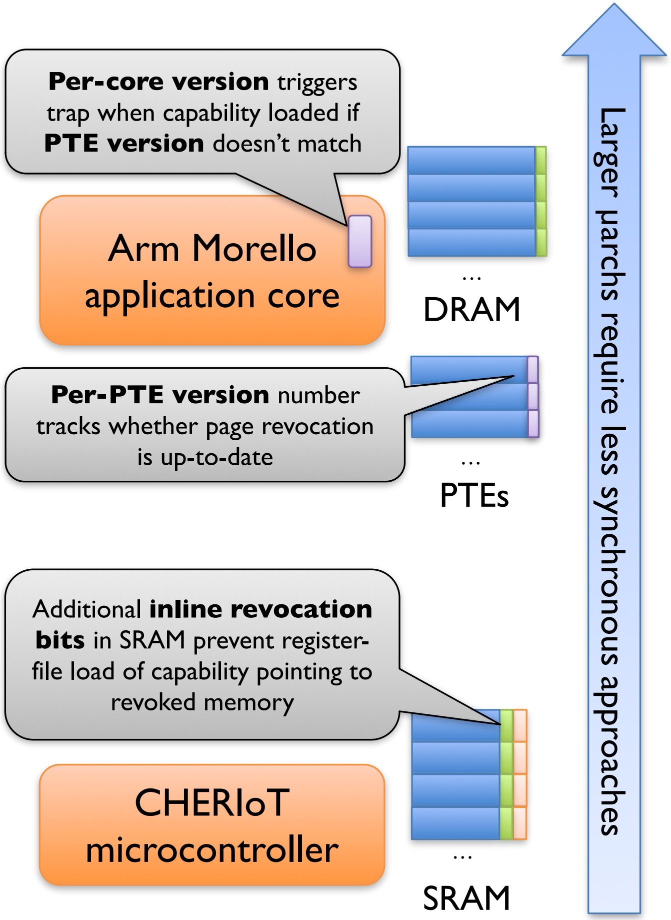
- Protections are applied to all pointer types in compiled code and the run-time environment:
  - **Implicit sub-language pointers** such as GOT entries, stack pointers, return addresses
  - **Explicit language-level pointers** to globals, stack and heap allocations, functions
- Software TCBs refine **bounds** and **permissions** during execution
  - E.g., the mmap() system call, run-time linker, heap allocator, and stack allocator

Implementing strong C/C++ memory protection with CHERI

Property	Architectural representation	Software invariants
Spatial safety	• Pointers implemented as capabilities • Bounds, permissions refined by allocators and linkers to describe spatial constraints	• Allocators and linkers maintain spatial mutual exclusion for all data types • Alignment and padding for imprecise bounds • free() uses rederivation if it relies on out-of-bounds accesses to reach metadata
Control-flow protection	• Control-flow pointers implemented as capabilities • Bounds, permissions refined by linkers, JITs to describe spatial constraints • Sentry capabilities prevent mutation and dereference of pointers when not in PCC	• Linkers, JITs maintains spatial mutual exclusion for all code pointers • Alignment and padding for imprecise bounds • Forward, backward, and exception edges protected • Higher-level policies such as W^X as desired
Temporal safety	• Load-barrier technique based on inline revocation bits (M-class cores) or CSR/PTE version bits (A-class cores) to implement atomicity, accelerate revocation • Tag stripped on revocation	• Allocators, linkers, and JITs maintain temporal mutual exclusion • Quarantine mechanism prevents reallocation before revocation, with strong atomicity

CHERI temporal safety

- Spatial safety maps naturally into capabilities … how about temporal safety?
  - The **use-after-free problem** maps into **capability revocation problem**
  - Tags enable **precise discovery** and **atomic revocation** of capabilities
  - Avoid architectural/microarchitectural indirection; instead use **sweeping revocation** to reliably find + invalidate capabilities
  - **Architectural load-barriers** defer and amortize sweeping
- Prevent **heap use-after-reallocation**
  - free() **quarantines** freed memory to defer, amortize sweeping costs
  - Invariants guarantee no valid pointers to freed memory before reallocating
- Sweeps provide "snapshot at the start"-like view of revocation:
  - In **microcontrollers**, checks can be synchronous (e.g., CHERIoT): Use **inline revocation bit** to check capabilities on load; clear tag if revoked
  - In **application cores**, operate page at a time (e.g., Arm Morello): Use **per-page version** to trap on capability loads if may contain revoked pointers, reducing pause times and enabling lazy sweeping techniques



Per-core version triggers trap when capability loaded if **PTE version** doesn't match

Arm Morello application core

…

DRAM

Per-PTE version number tracks whether page revocation is up-to-date

…

PTEs

Additional **inline revocation bits** in SRAM prevent register-file load of capability pointing to revoked memory

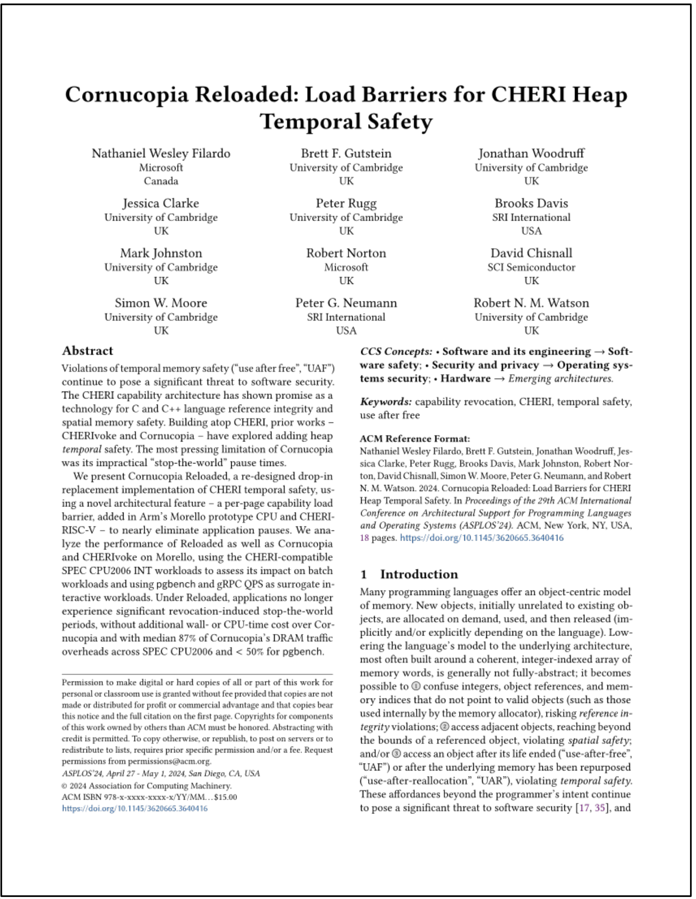
CHERIoT microcontroller

…

SRAM

Larger µarchs require less synchronous approaches

Cornucopia Reloaded: Load barriers (ASPLOS 2024)



Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety

Nathaniel Wesley Filardo
Microsoft
Canada

Brett F. Gutstein
University of Cambridge
UK

Jonathan Woodruff
University of Cambridge
UK

Jessica Clarke
University of Cambridge
UK

Peter Rugg
University of Cambridge
UK

Brooks Davis
SRI International
USA

Mark Johnston
University of Cambridge
UK

Robert Norton
Microsoft
UK

David Chisnall
SCI Semiconductor
UK

Simon W. Moore
University of Cambridge
UK

Peter G. Neumann
SRI International
USA

Robert N. M. Watson
University of Cambridge
UK

Abstract

Violations of temporal memory safety ("use after free", "UAF") continue to pose a significant threat to software security. The CHERI capability architecture has shown promise as a technology for C and C++ language reference integrity and spatial memory safety. Building atop CHERI, prior works – CHERIvoke and Cornucopia – have explored adding heap *temporal* safety. The most pressing limitation of Cornucopia was its impractical "stop-the-world" pause times.

We present Cornucopia Reloaded, a re-designed drop-in replacement implementation of CHERI temporal safety, using a novel architectural feature – a per-page capability load barrier, added in Arm's Morello prototype CPU and CHERI-RISC-V – to nearly eliminate application pauses. We analyze the performance of Reloaded as well as Cornucopia and CHERIvoke on Morello, using the CHERI-compatible SPEC CPU2006 INT workloads to assess its impact on batch workloads and using pgbench and gRPC QPS as surrogate interactive workloads. Under Reloaded, applications no longer experience significant revocation-induced stop-the-world periods, without additional wall- or CPU-time cost over Cornucopia and with median 87% of Cornucopia's DRAM traffic overheads across SPEC CPU2006 and < 50% for pgbench.

ASPLOS'24, April 27 - May 1, 2024, San Diego, CA, USA
© 2024 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/3620665.3640416

CCS Concepts: • **Software and its engineering** → **Software safety**; • **Security and privacy** → **Operating systems security**; • **Hardware** → *Emerging architectures.*

Keywords: capability revocation, CHERI, temporal safety, use after free

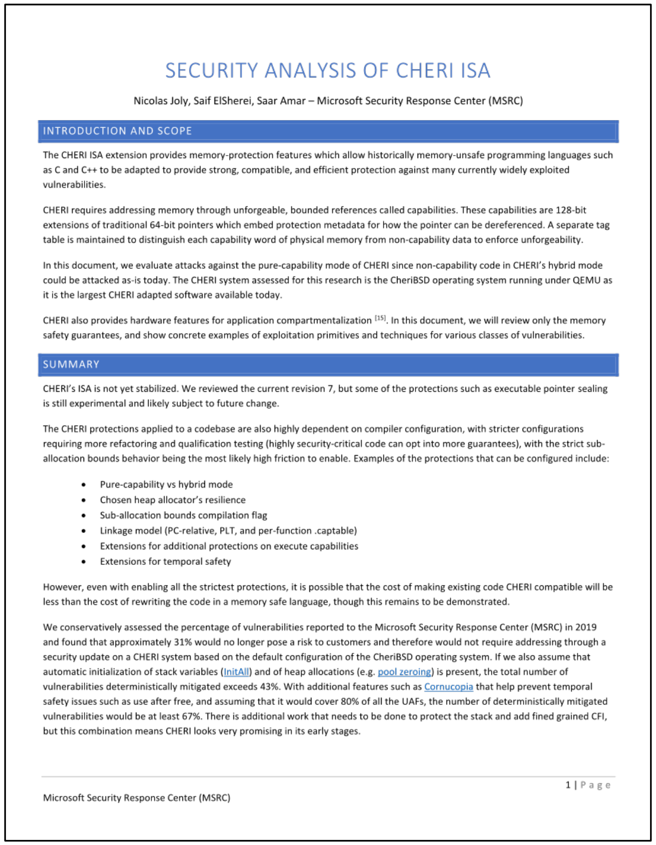
ACM Reference Format:
Nathaniel Wesley Filardo, Brett F. Gutstein, Jonathan Woodruff, Jessica Clarke, Peter Rugg, Brooks Davis, Mark Johnston, Robert Norton, David Chisnall, Simon W. Moore, Peter G. Neumann, and Robert N. M. Watson. 2024. Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24).* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3620665.3640416

1 Introduction

Many programming languages offer an object-centric model of memory. New objects, initially unrelated to existing objects, are allocated on demand, used, and then released (implicitly and/or explicitly depending on the language). Lowering the language's model to the underlying architecture, most often built around a coherent, integer-indexed array of memory words, is generally not fully-abstract; it becomes possible to ① confuse integers, object references, and memory indices that do not point to valid objects (such as those used internally by the memory allocator), risking *reference integrity* violations; ② access adjacent objects, reaching beyond the bounds of a referenced object, violating *spatial safety*; and/or ③ access an object after its life ended ("use-after-free", "UAF") or after the underlying memory has been repurposed ("use-after-reallocation", "UAR"), violating *temporal safety*. These affordances beyond the programmer's intent continue to pose a significant threat to software security [17, 35], and

- **Cornucopia heap temporal safety** (IEEE SSP 2020), is a GC-inspired, quarantining technique
  - The kernel VM system tracks "capability dirty" pages
  - A long "stop-the-world" phase - as much as 30 milliseconds measured in practice
- **Cornucopia Reloaded** (ASPLOS 2024) moves to a GC-inspired "load-barrier"
  - VM invariant is that accessible pages have already undergone revocation
  - Depend on 1-bit capability generation added to VM PTEs, implemented by Morello
  - Stop-the-world pauses tens of microseconds
- Enabled by default since CheriBSD 23.11

Microsoft security analysis of CHERI C/C++



SECURITY ANALYSIS OF CHERI ISA

Nicolas Joly, Saif ElSherei, Saar Amar – Microsoft Security Response Center (MSRC)

INTRODUCTION AND SCOPE

The CHERI ISA extension provides memory-protection features which allow historically memory-unsafe programming languages such as C and C++ to be adapted to provide strong, compatible, and efficient protection against many currently widely exploited vulnerabilities.

CHERI requires addressing memory through unforgeable, bounded references called capabilities. These capabilities are 128-bit extensions of traditional 64-bit pointers which embed protection metadata for how the pointer can be dereferenced. A separate tag table is maintained to distinguish each capability word of physical memory from non-capability data to enforce unforgeability.

In this document, we evaluate attacks against the pure-capability mode of CHERI since non-capability code in CHERI's hybrid mode could be attacked as-is today. The CHERI system assessed for this research is the CheriBSD operating system running under QEMU as it is the largest CHERI adapted software available today.

CHERI also provides hardware features for application compartmentalization [15]. In this document, we will review only the memory safety guarantees, and show concrete examples of exploitation primitives and techniques for various classes of vulnerabilities.

SUMMARY

CHERI's ISA is not yet stabilized. We reviewed the current revision 7, but some of the protections such as executable pointer sealing is still experimental and likely subject to future change.

The CHERI protections applied to a codebase are also highly dependent on compiler configuration, with stricter configurations requiring more refactoring and qualification testing (highly security-critical code can opt into more guarantees), with the strict sub-allocation bounds behavior being the most likely high friction to enable. Examples of the protections that can be configured include:

- Pure-capability vs hybrid mode
- Chosen heap allocator's resilience
- Sub-allocation bounds compilation flag
- Linkage model (PC-relative, PLT, and per-function .captable)
- Extensions for additional protections on execute capabilities
- Extensions for temporal safety

However, even with enabling all the strictest protections, it is possible that the cost of making existing code CHERI compatible will be less than the cost of rewriting the code in a memory safe language, though this remains to be demonstrated.

We conservatively assessed the percentage of vulnerabilities reported to the Microsoft Security Response Center (MSRC) in 2019 and found that approximately 31% would no longer pose a risk to customers and therefore would not require addressing through a security update on a CHERI system based on the default configuration of the CheriBSD operating system. If we also assume that automatic initialization of stack variables (InitAll) and of heap allocations (e.g. pool zeroing) is present, the total number of vulnerabilities deterministically mitigated exceeds 43%. With additional features such as Cornucopia that help prevent temporal safety issues such as use after free, and assuming that it would cover 80% of all the UAFs, the number of deterministically mitigated vulnerabilities would be at least 67%. There is additional work that needs to be done to protect the stack and add fined grained CFI, but this combination means CHERI looks very promising in its early stages.

1 | Page

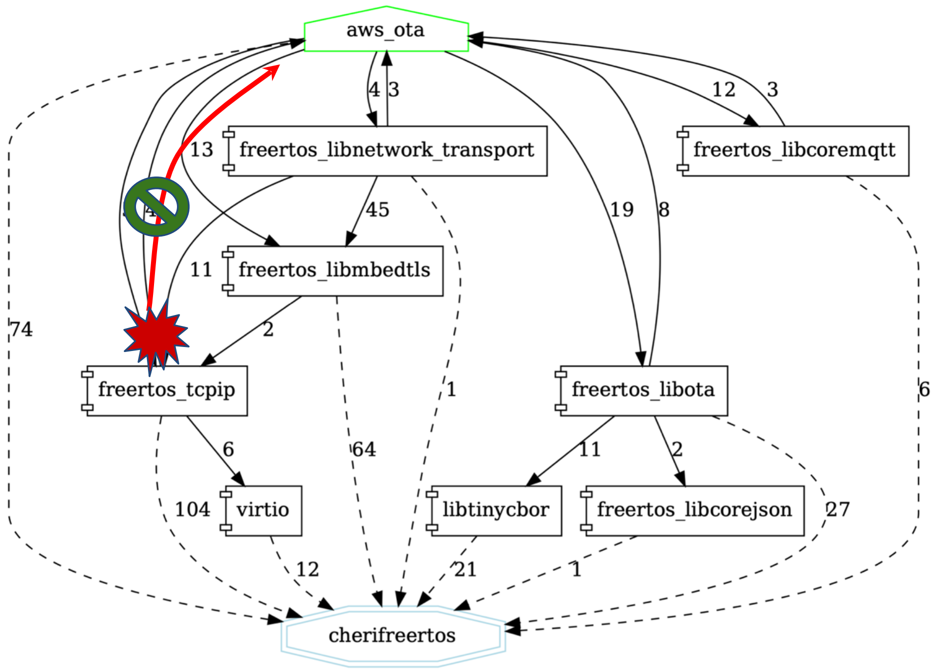
Microsoft Security Response Center (MSRC)

- Microsoft Security Response Center (MSRC) study analyzed all 2019 Microsoft critical memory-safety security vulnerabilities
  - Metric: "Poses a risk to customers → requires a software update"
  - Vulnerability mitigated if **no security update required**
- Blog post and 42-page report
  - Concrete vulnerability analysis for spatial safety
  - Abstract analysis of the impact of temporal safety
  - Red teaming of specific artifacts to gain experience
- CHERI, "in its current state, and combined with other mitigations, it would have **deterministically mitigated at least two thirds of all those issues**"

https://msrc-blog.microsoft.com/2020/10/14/security-analysis-of-cheri-isa/

CHERI SOFTWARE COMPARTMENTALISATION

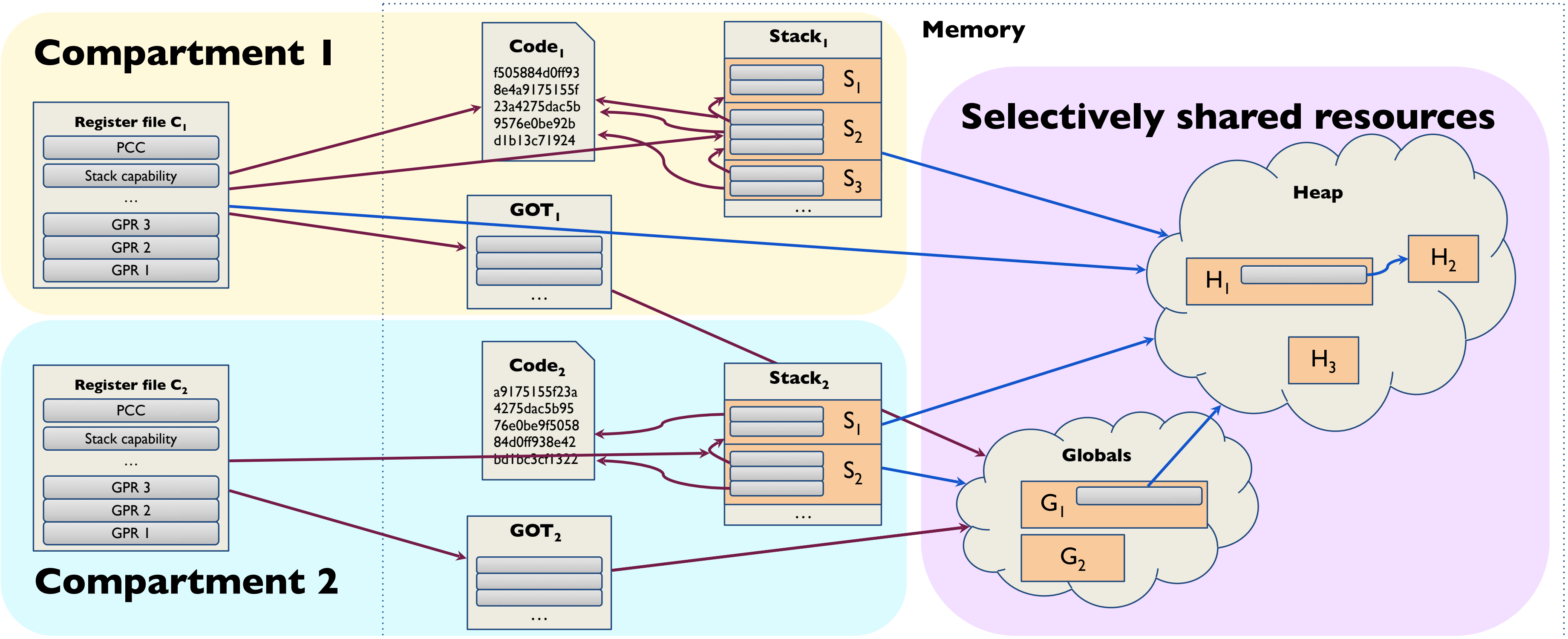
What is software compartmentalization?



CheriFreeRTOS components and the application execute in compartments. CHERI contains an attack within TCP/IP compartment, which access neither flash nor the internals of the software update (OTA) compartment.

- Fine-grained decomposition of a larger software system into **isolated modules** to constrain the impact of faults or attacks
- Goals is to **minimize privileges yielded by a successful attack, and to limit further attack surfaces**
- Usefully thought about as a **graph of interconnected components**, where the attacker's goal is to compromise nodes of the graph providing a route from a point of entry to a specific target

Compartmentalization built on CHERI memory safety



- Software components can be strongly isolated from one another within a shared address space
- Initial sharing is authorized by the TCB (functions, globals, …) by delegating capabilities
- Fast switching, shared memory w/o TLB contention, capability delegation accelerate sharing

Compartmentalization scalability

- CHERI dramatically improves **compartmentalization scalability**
  - More compartments
  - More frequent and faster domain transitions
  - Faster shared memory between compartments
- Many potential use cases – e.g., sandbox processing of each image in a web browser, processing each message in a mail application

Early benchmarks show a 1-to-2 order of magnitude performance inter-compartment communication improvement compared to conventional designs

Two promising CHERI-based compartmentalization models

Library compartmentalization (c18n)

- Reuse existing library (or library subset) abstractions
- Existing memory-safe linkage as foundation
- Interpose domain-transition trampolines on PLT calls
- Separate per-compartment stacks
- Policy language enables sub-library boundaries to be used

Shipped in CheriBSD 22.12.

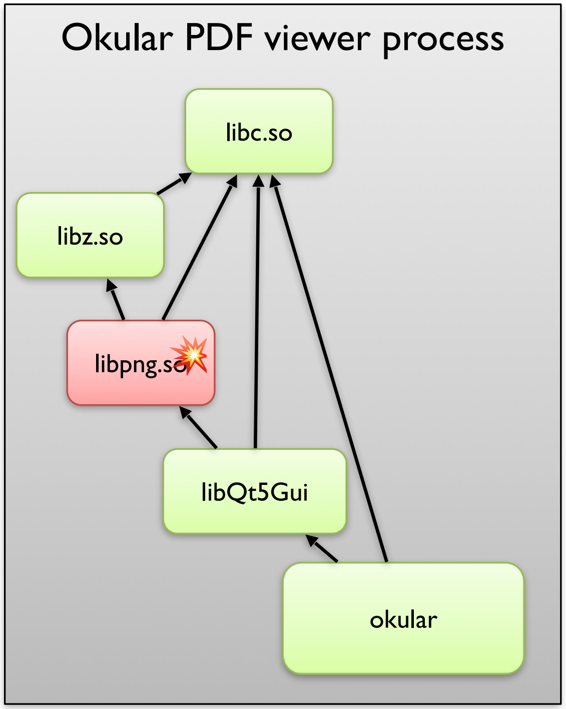
Updates in 23.11, 24.05, and 25.03.

UNIX co-located processes (co-processes)

- Multiple processes in each address space, separate by CHERI
- Kernel protrudes fast domain-transition switcher into userlevel
- Inter-process shared memory shares TLB entries
- Measured 1-2 orders of magnitude performance vs. IPC

Prototype running; targeted for inclusion in a 2026 CheriBSD release.

CHERI library compartmentalization (c18n)



Libraries are software-engineering boundaries around code

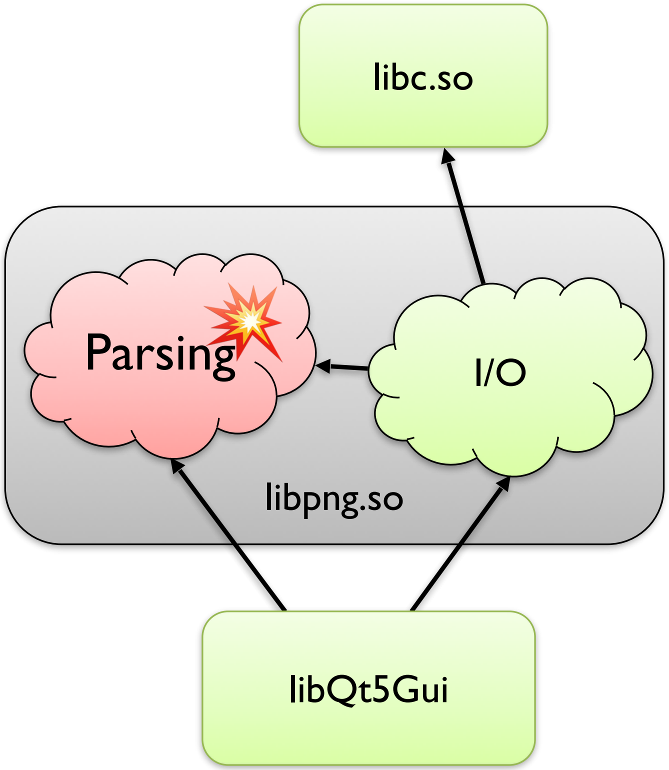
- E.g., libz (decompression) and libpng (PNG processing)

Compartmentalization limits access to global state, control flow:

- **Control-flow protection** for cross-compartment calls
- **Granular delegation of access** to global variables and functions from other libraries based on link-time policy
- Prevents **direct use of system calls** and kernel services
- Prevents integrity, confidentiality, and control-flow cross-compartment **attacks via stacks**
- Dynamically grants further rights to resources (e.g., heap allocations, function pointers, …) via **pointer delegation**

Support for library compartmentalization has shipped in multiple CheriBSD releases and is experimental but very usable

CHERI sub-library compartmentalization



However, sometimes **software engineering boundaries aren’t the right security boundaries**; e.g.,

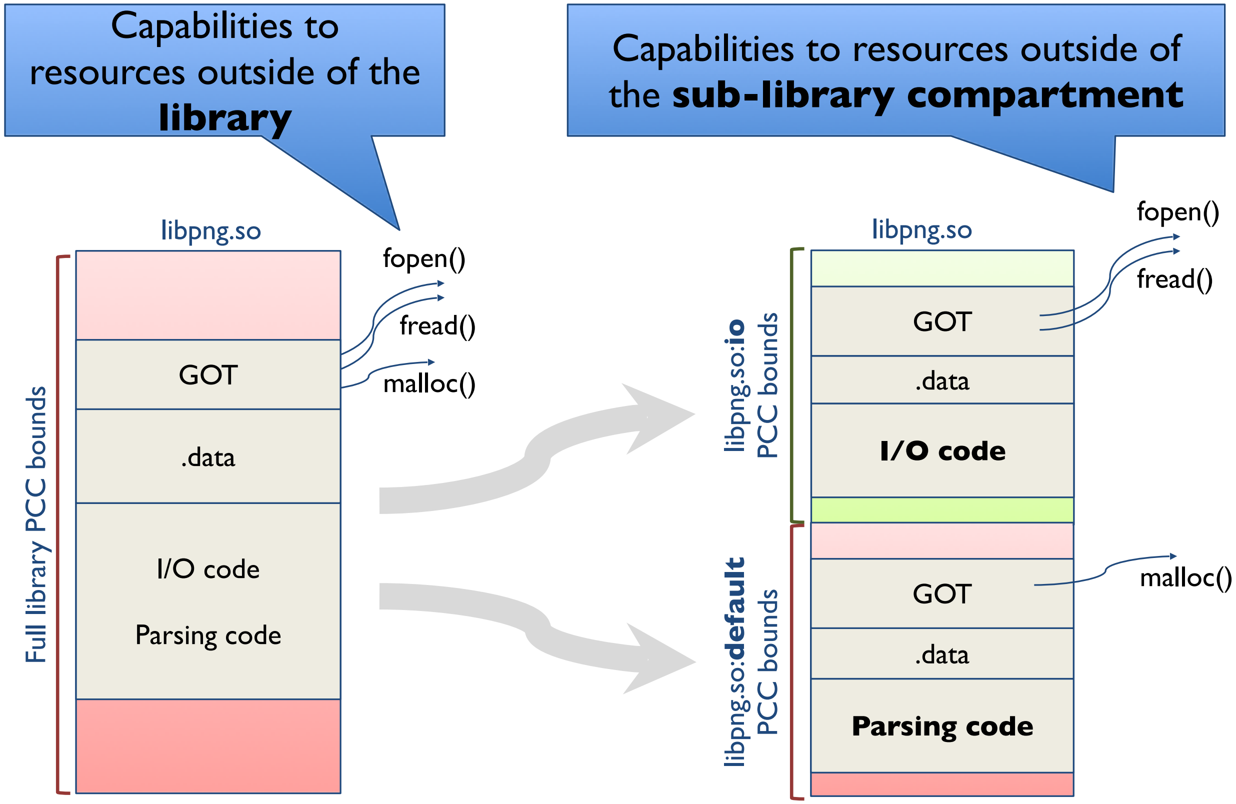
- libpng is vulnerable when processing malicious data …
- … but some libpng APIs requires file-system access

Sub-library compartmentalization allows compartment boundaries to differ from library boundaries without refactoring

- Subdivide library state, library code, and access requirements into multiple compartments within library
- Compromise in one component in a library (e.g., image parsing) need not imply compromise of another (e.g., I/O)

Experimental feature in April 2025 CheriBSD release

Sub-library compartment approach



Capabilities to resources outside of the **library**
libpng.so
Full library PCC bounds
GOT
.data
I/O code
Parsing code
fopen()
fread()
malloc()
Capabilities to resources outside of the **sub-library compartment**
libpng.so
libpng.so:**io** PCC bounds
GOT
.data
I/O code
fopen()
fread()
libpng.so:**default** PCC bounds
GOT
.data
Parsing code
malloc()

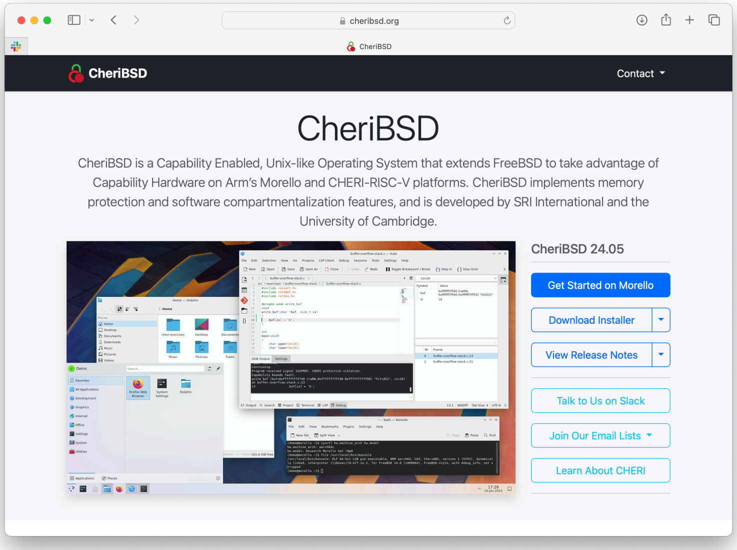
Sub-library compartments are contiguous code, data, and linkage that can access only authorized resources:

- New ELF metadata specifies compartment details including name and **Program-Counter Capability (PCC)** bounds
- External functions and globals are reached via per-compartment **Global Offset Tables (GOTs)**, reached via each compartment's PCC

The runtime also applies appropriate stack isolation, control-flow, and other protections to sub-library compartments

OPERATING-SYSTEM SUPPORT: CHERIBSD RESEARCH OS

CheriBSD research operating system



CheriBSD

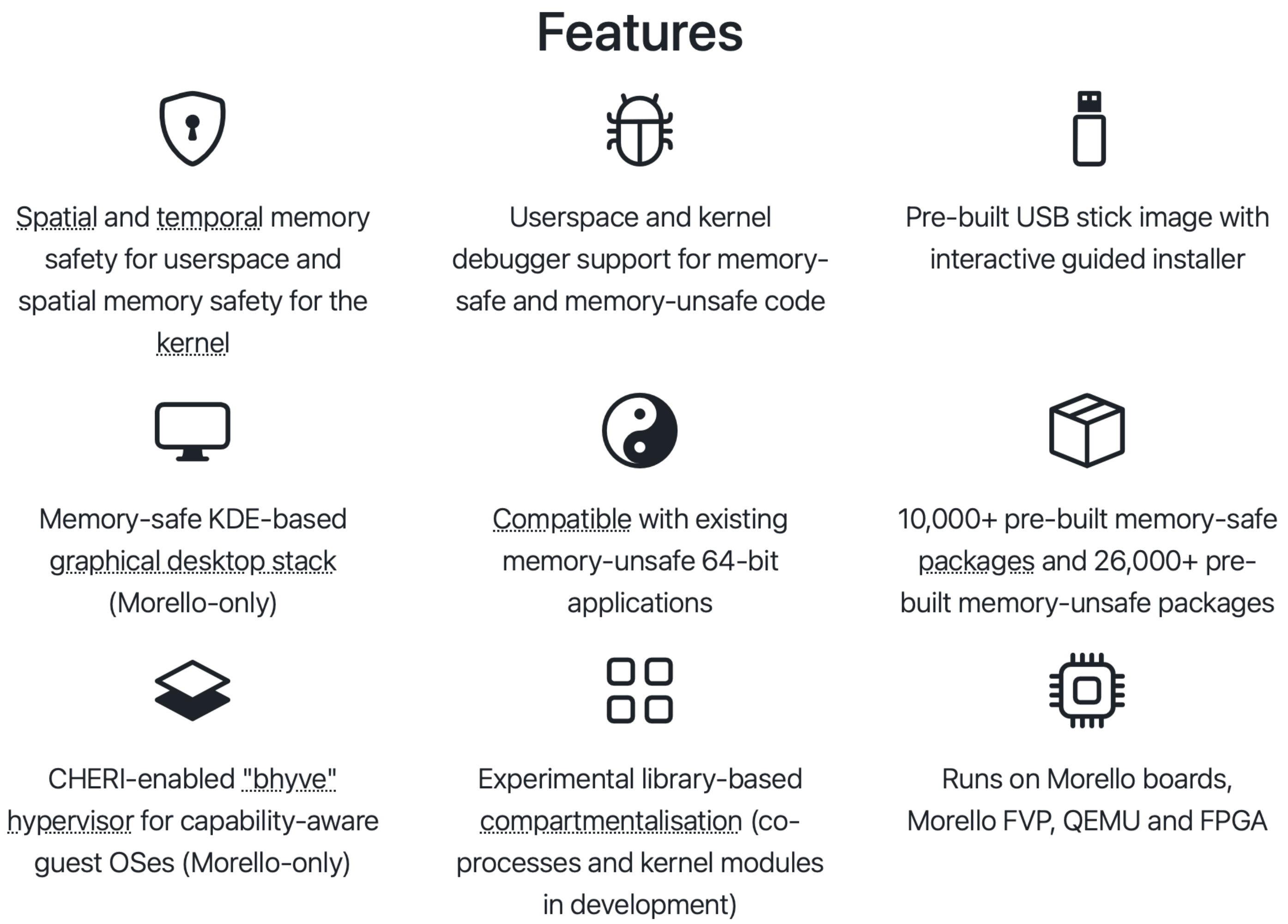
CheriBSD is a Capability Enabled, Unix-like Operating System that extends FreeBSD to take advantage of Capability Hardware on Arm's Morello and CHERI-RISC-V platforms. CheriBSD implements memory protection and software compartmentalization features, and is developed by SRI International and the University of Cambridge.

CheriBSD 24.05

CheriBSD is a FreeBSD-based research operating system designed to explore OS use of CHERI:

- **Incrementally deployed** use of CHERI: Not a clean-slate OS design!
- **Universal use of CHERI**: Ubiquitous memory safety and compartmentalization
- **14 years of hardware-software co-design** alongside CHERI architecture and the CHERI compiler, driving critical design choices throughout CHERI architecture and microarchitecture
- **New OS design principles** that themselves are systems research contributions, and are generalizable to other systems (e.g., CHERI-seL4 and CHERI Linux being developed currently)

https://cheribsd.org



Features

Spatial and temporal memory safety for userspace and spatial memory safety for the kernel

Userspace and kernel debugger support for memory-safe and memory-unsafe code

Pre-built USB stick image with interactive guided installer

Memory-safe KDE-based graphical desktop stack (Morello-only)

Compatible with existing memory-unsafe 64-bit applications

10,000+ pre-built memory-safe packages and 26,000+ pre-built memory-unsafe packages

CHERI-enabled "bhyve" hypervisor for capability-aware guest OSes (Morello-only)

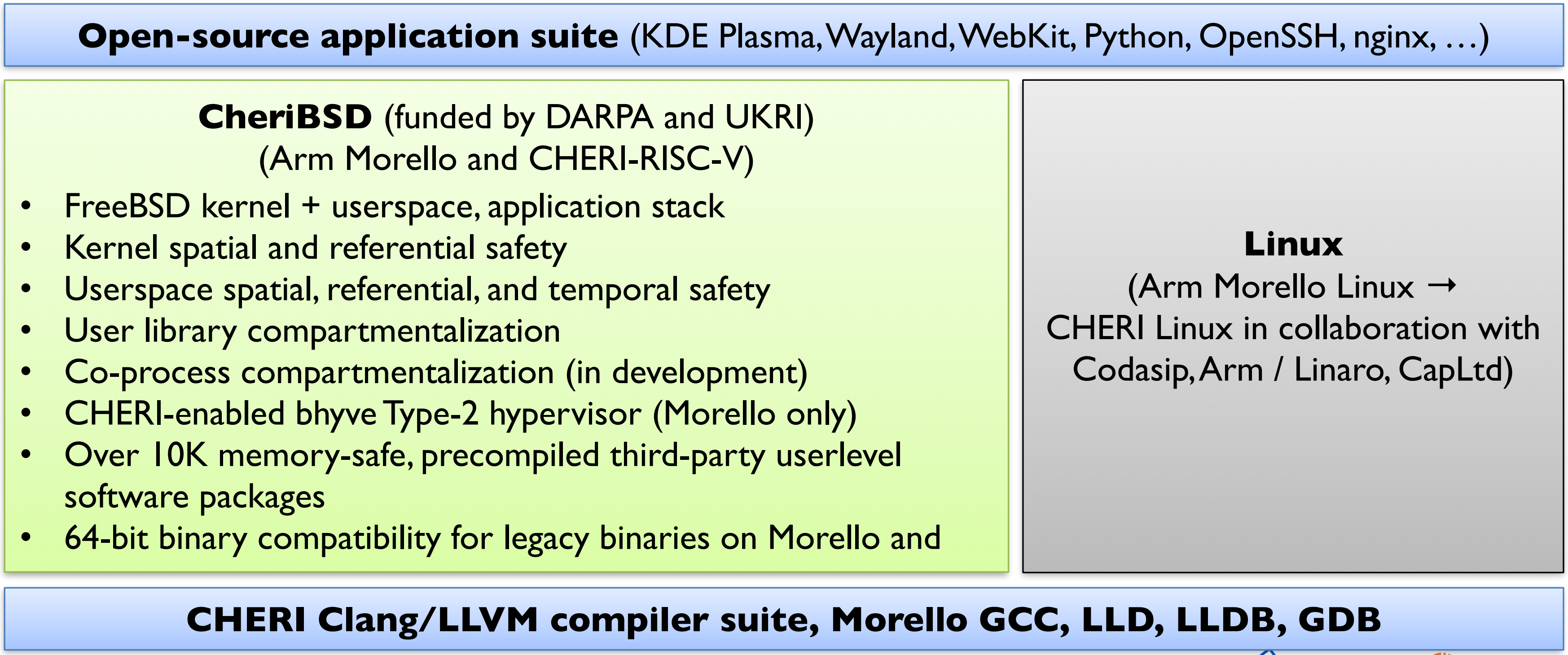
Experimental library-based compartmentalisation (co-processes and kernel modules in development)

Runs on Morello boards, Morello FVP, QEMU and FPGA

A PROTOTYPE SOFTWARE ECOSYSTEM FOR DEMONSTRATION AND EVALUATION

CHERI prototype software stack

Complete CHERI-enabled open-source software stack from bare metal up: compilers/toolchain, debugger, hypervisor, OS, and applications to evaluate, demonstrate 100+ MLoC of memory-safe CHERI C/C++ with incremental adoption path:



Open-source application suite (KDE Plasma, Wayland, WebKit, Python, OpenSSH, nginx, …)

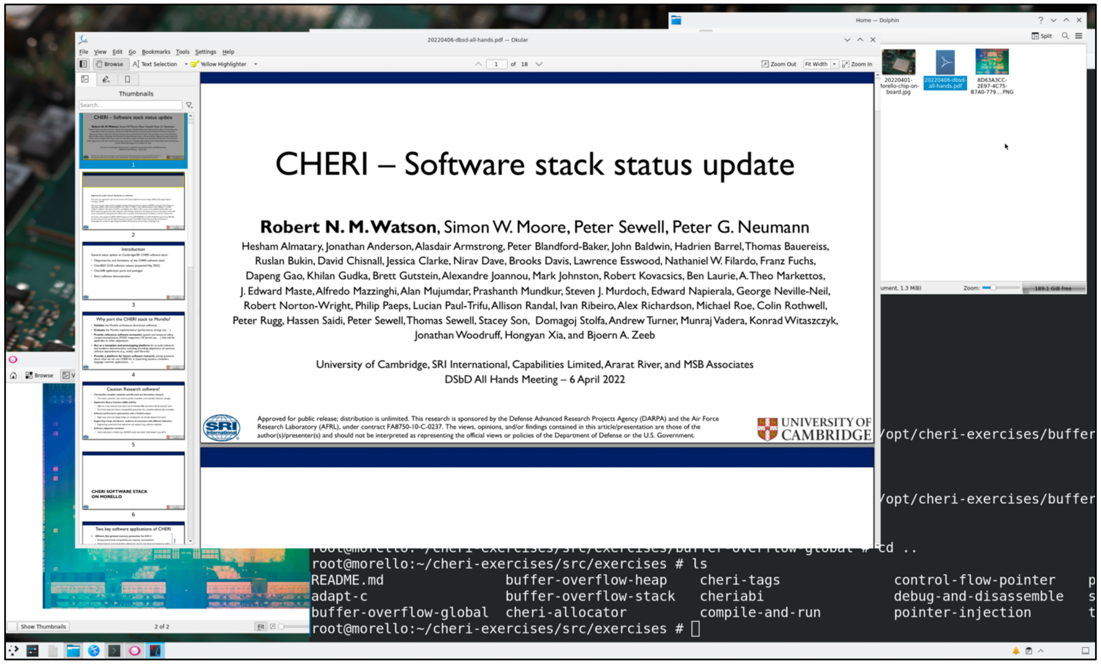
CheriBSD (funded by DARPA and UKRI)
(Arm Morello and CHERI-RISC-V)

- FreeBSD kernel + userspace, application stack
- Kernel spatial and referential safety
- Userspace spatial, referential, and temporal safety
- User library compartmentalization
- Co-process compartmentalization (in development)
- CHERI-enabled bhyve Type-2 hypervisor (Morello only)
- Over 10K memory-safe, precompiled third-party userlevel software packages
- 64-bit binary compatibility for legacy binaries on Morello and

Linux
(Arm Morello Linux →
CHERI Linux in collaboration with
Codasip, Arm / Linaro, CapLtd)

CHERI Clang/LLVM compiler suite, Morello GCC, LLD, LLDB, GDB

2021 desktop pilot study results


20220406-dbsd-all-hands.pdf — Okular
Home — Dolphin
Thumbnails
Yellow Highlighter
Zoom Out
Fit Width
Zoom In
1 of 18
20220401-morello-chip-on-board.jpg
20220406-dbsd-all-hands.pdf
Show Thumbnails
2 of 2

CHERI – Software stack status update

Robert N. M. Watson, Simon W. Moore, Peter Sewell, Peter G. Neumann
Hesham Almatary, Jonathan Anderson, Alasdair Armstrong, Peter Blandford-Baker, John Baldwin, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, David Chisnall, Jessica Clarke, Nirav Dave, Brooks Davis, Lawrence Esswood, Nathaniel W. Filardo, Franz Fuchs, Dapeng Gao, Khilan Gudka, Brett Gutstein, Alexandre Joannou, Mark Johnston, Robert Kovacsics, Ben Laurie, A. Theo Markettos, J. Edward Maste, Alfredo Mazzinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, George Neville-Neil, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Allison Randal, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Peter Sewell, Thomas Sewell, Stacey Son, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Jonathan Woodruff, Hongyan Xia, and Bjoern A. Zeeb

University of Cambridge, SRI International, Capabilities Limited, Ararat River, and MSB Associates
DSbD All Hands Meeting – 6 April 2022

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237. The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

```
/opt/cheri-exercises/buffer

/opt/cheri-exercises/buffer
root@morello:~/cheri-exercises/src/exercises/buffer-overflow-global # cd ..
root@morello:~/cheri-exercises/src/exercises # ls
README.md               buffer-overflow-heap    cheri-tags              control-flow-pointer    p
adapt-c                 buffer-overflow-stack   cheriabi                debug-and-disassemble   s
buffer-overflow-global  cheri-allocator         compile-and-run         pointer-injection       t
root@morello:~/cheri-exercises/src/exercises # 
```

Developed:

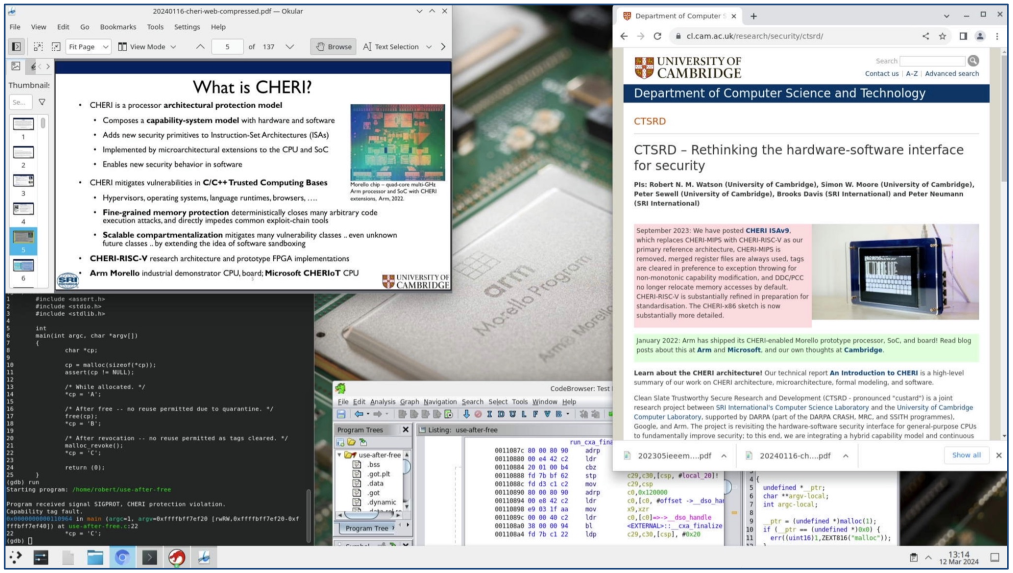
- **6 million lines of C/C++ code** compiled for memory safety; modest dynamic testing
- **Three compartmentalization whiteboard case studies** in Qt/KDE

Evaluation results:

- **0.026% LoC modification rate** across full corpus for memory safety
- **73.8% mitigation rate** across full corpus, using memory safety and compartmentalization

Useful observation to be made about memory safety: Not enough to address the de facto threat model of quite a few libraries …

2024.05 Morello memory-safe desktop stack



20240116-cheri-web-compressed.pdf — Okular
File View Edit Go Bookmarks Tools Settings Help
Fit Page
View Mode
5
of 137
Browse
Text Selection
Thumbnails
Se...
1
2
3
4
5
6

What is CHERI?

- CHERI is a processor **architectural protection model**
  - Composes a **capability-system model** with hardware and software
  - Adds new security primitives to Instruction-Set Architectures (ISAs)
  - Implemented by microarchitectural extensions to the CPU and SoC
  - Enables new security behavior in software
- CHERI mitigates vulnerabilities in **C/C++ Trusted Computing Bases**
  - Hypervisors, operating systems, language runtimes, browsers, ….
  - **Fine-grained memory protection** deterministically closes many arbitrary code execution attacks, and directly impedes common exploit-chain tools
  - **Scalable compartmentalization** mitigates many vulnerability classes .. even unknown future classes .. by extending the idea of software sandboxing
- **CHERI-RISC-V** research architecture and prototype FPGA implementations
- **Arm Morello** industrial demonstrator CPU, board; **Microsoft CHERIoT** CPU

Morello chip – quad-core multi-GHz Arm processor and SoC with CHERI extensions, Arm, 2022.

Department of Computer S
cl.cam.ac.uk/research/security/ctsrd/
UNIVERSITY OF CAMBRIDGE
Search
Contact us | A-Z | Advanced search

Department of Computer Science and Technology

CTSRD

CTSRD – Rethinking the hardware-software interface for security

PIs: Robert N. M. Watson (University of Cambridge), Simon W. Moore (University of Cambridge), Peter Sewell (University of Cambridge), Brooks Davis (SRI International) and Peter Neumann (SRI International)

September 2023: We have posted **CHERI ISAv9**, which replaces CHERI-MIPS with CHERI-RISC-V as our primary reference architecture, CHERI-MIPS is removed, merged register files are always used, tags are cleared in preference to exception throwing for non-monotonic capability modification, and DDC/PCC no longer relocate memory accesses by default. CHERI-RISC-V is substantially refined in preparation for standardisation. The CHERI-x86 sketch is now substantially more detailed.

January 2022: Arm has shipped its CHERI-enabled Morello prototype processor, SoC, and board! Read blog posts about this at **Arm** and **Microsoft**, and our own thoughts at **Cambridge**.

Learn about the CHERI architecture! Our technical report **An Introduction to CHERI** is a high-level summary of our work on CHERI architecture, microarchitecture, formal modeling, and software.

Clean Slate Trustworthy Secure Research and Development (CTSRD - pronounced "custard") is a joint research project between SRI International's Computer Science Laboratory and the University of Cambridge Computer Laboratory, supported by DARPA (part of the DARPA CRASH, MRC, and SSITH programmes), Google, and Arm. The project is revisiting the hardware-software security interface for general-purpose CPUs to fundamentally improve security; to this end, we are integrating a hybrid capability model and continuous

```
1       #include <assert.h>
2       #include <stdio.h>
3       #include <stdlib.h>
4
5       int
6       main(int argc, char *argv[])
7       {
8               char *cp;
9
10              cp = malloc(sizeof(*cp));
11              assert(cp != NULL);
12
13              /* While allocated. */
14              *cp = 'A';
15
16              /* After free -- no reuse permitted due to quarantine. */
17              free(cp);
18              *cp = 'B';
19
20              /* After revocation -- no reuse permitted as tags cleared. */
21              malloc_revoke();
22              *cp = 'C';
23
24              return (0);
25      }
(gdb) run
Starting program: /home/robert/use-after-free

Program received signal SIGPROT, CHERI protection violation.
Capability tag fault.
0x0000000000110964 in main (argc=1, argv=0xffffbff7ef20 [rwRW,0xffffbff7ef20-0xffffbff7ef40]) at use-after-free.c:22
22              *cp = 'C';
(gdb) 
```

```
                         run_cxa_fina
0011087c 80 00 80 90     adrp
00110880 00 e4 42 c2     ldr
00110884 20 01 00 b4     cbz
00110888 fd 7b bf 62     stp      c29,c30,[csp, #local_20]!
0011088c fd d3 c1 c2     mov      c29,csp
00110890 80 00 80 90     adrp     c0,0x120000
00110894 00 e8 42 c2     ldr      c0,[c0, #offset ->__dso_han
00110898 e9 03 1f aa     mov      x9,xzr
0011089c 00 00 40 c2     ldr      c0,[c0]=>->__dso_handle
001108a0 38 00 00 94     bl       <EXTERNAL>::__cxa_finalize
001108a4 fd 7b c1 22     ldp      c29,c30,[csp], #0x20
```

```
{
  undefined *__ptr;
  char **argv-local;
  int argc-local;

  __ptr = (undefined *)malloc(1);
  if (__ptr == (undefined *)0x0) {
    err((uint16)1,ZEXT816("malloc"));
```

50-100MLoC of memory-safe C/C++ on a shipping Arm Morello prototype board today:

- CheriBSD kernel with DRM + Panfrost drivers
- CheriBSD userspace with libraries and tools
- Plasma, KDE base applications including Dolphin, Okular, Kate, and Konsole
- Library compartmentalization of all memory-safe userlevel components
- Rich software development environment including Clang/LLVM, GDB, Ghidra, …
- Roughly 10K memory-safe third-party software packages, and 20K aarch64 packages
- Also includes memory-safe server software such as gRPC, nginx, postgres, …

Some more complex, un-adapted applications (e.g., Chromium, OpenJDK) run in 64-bit Arm mode

CHERI and legacy applications side-by-side on Morello

Enables incremental application migration to memory safety

Memory-safe PDF viewer

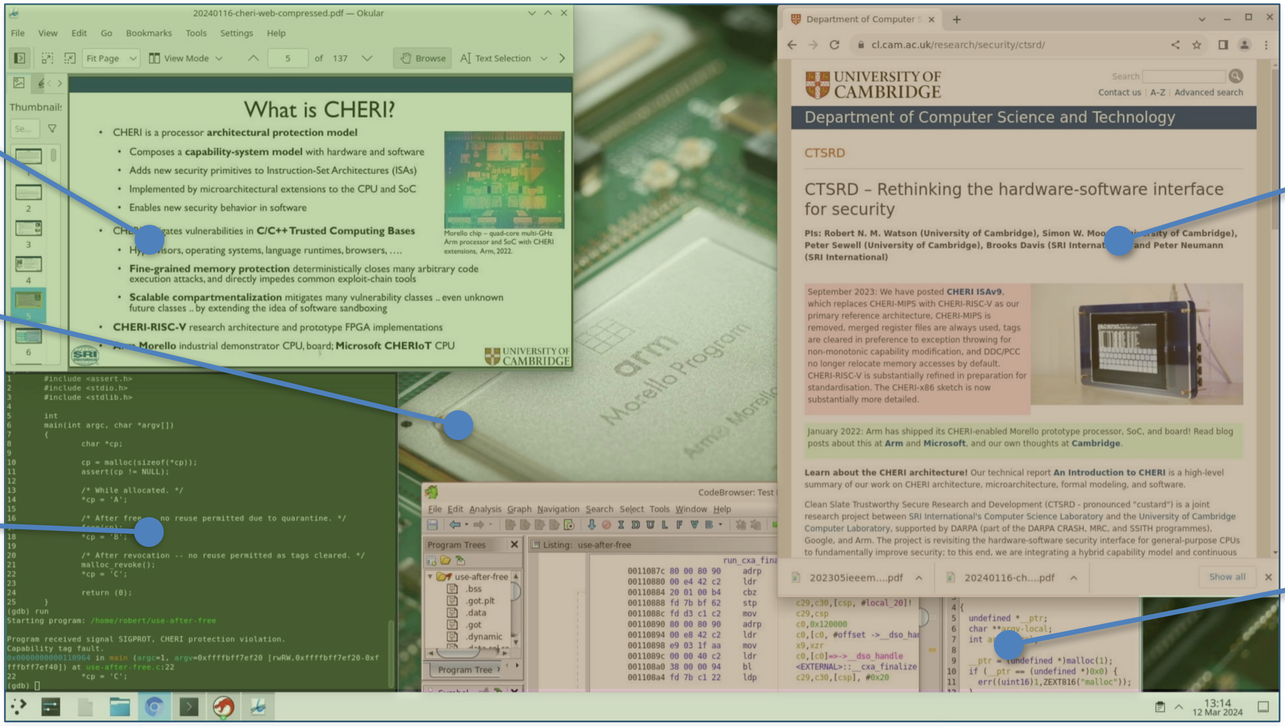
Memory-safe desktop environment

Memory-safe terminal window and commands

Memory-safe OS kernel

Legacy 64-bit Arm Chromium browser

Legacy 64-bit Arm JVM



20240116-cheri-web-compressed.pdf — Okular
File View Edit Go Bookmarks Tools Settings Help
Fit Page
View Mode
5 of 137
Browse
Text Selection
Thumbnails
1
2
3
4
5
6

What is CHERI?

- CHERI is a processor **architectural protection model**
  - Composes a **capability-system model** with hardware and software
  - Adds new security primitives to Instruction-Set Architectures (ISAs)
  - Implemented by microarchitectural extensions to the CPU and SoC
  - Enables new security behavior in software
- CHERI mitigates vulnerabilities in **C/C++ Trusted Computing Bases**
  - Hypervisors, operating systems, language runtimes, browsers, ….
  - **Fine-grained memory protection** deterministically closes many arbitrary code execution attacks, and directly impedes common exploit-chain tools
  - **Scalable compartmentalization** mitigates many vulnerability classes .. even unknown future classes .. by extending the idea of software sandboxing
- **CHERI-RISC-V** research architecture and prototype FPGA implementations
- **Arm Morello** industrial demonstrator CPU, board; **Microsoft CHERIoT** CPU

Morello chip – quad-core multi-GHz Arm processor and SoC with CHERI extensions, Arm, 2022.

```
1       #include <assert.h>
2       #include <stdio.h>
3       #include <stdlib.h>
4
5       int
6       main(int argc, char *argv[])
7       {
8               char *cp;
9
10              cp = malloc(sizeof(*cp));
11              assert(cp != NULL);
12
13              /* While allocated. */
14              *cp = 'A';
15
16              /* After free -- no reuse permitted due to quarantine. */
17              free(cp);
18              *cp = 'B';
19
20              /* After revocation -- no reuse permitted as tags cleared. */
21              malloc_revoke();
22              *cp = 'C';
23
24              return (0);
25      }
(gdb) run
Starting program: /home/robert/use-after-free

Program received signal SIGPROT, CHERI protection violation.
Capability tag fault.
0x0000000000110964 in main (argc=1, argv=0xffffbff7ef20 [rwRW,0xffffbff7ef20-0xf
fffbff7ef40]) at use-after-free.c:22
22              *cp = 'C';
(gdb) 
```

Department of Computer S
cl.cam.ac.uk/research/security/ctsrd/
UNIVERSITY OF CAMBRIDGE
Search
Contact us | A-Z | Advanced search

Department of Computer Science and Technology

CTSRD

CTSRD – Rethinking the hardware-software interface for security

PIs: Robert N. M. Watson (University of Cambridge), Simon W. Moore (University of Cambridge), Peter Sewell (University of Cambridge), Brooks Davis (SRI International), and Peter Neumann (SRI International)

September 2023: We have posted **CHERI ISAv9**, which replaces CHERI-MIPS with CHERI-RISC-V as our primary reference architecture, CHERI-MIPS is removed, merged register files are always used, tags are cleared in preference to exception throwing for non-monotonic capability modification, and DDC/PCC no longer relocate memory accesses by default. CHERI-RISC-V is substantially refined in preparation for standardisation. The CHERI-x86 sketch is now substantially more detailed.

January 2022: Arm has shipped its CHERI-enabled Morello prototype processor, SoC, and board! Read blog posts about this at **Arm** and **Microsoft**, and our own thoughts at **Cambridge**.

Learn about the CHERI architecture! Our technical report **An Introduction to CHERI** is a high-level summary of our work on CHERI architecture, microarchitecture, formal modeling, and software.

Clean Slate Trustworthy Secure Research and Development (CTSRD - pronounced "custard") is a joint research project between SRI International's Computer Science Laboratory and the University of Cambridge Computer Laboratory, supported by DARPA (part of the DARPA CRASH, MRC, and SSITH programmes), Google, and Arm. The project is revisiting the hardware-software security interface for general-purpose CPUs to fundamentally improve security; to this end, we are integrating a hybrid capability model and continuous

```
                 run_cxa_fina
0011087c 80 00 80 90     adrp
00110880 00 e4 42 c2     ldr
00110884 20 01 00 b4     cbz
00110888 fd 7b bf 62     stp      c29,c30,[csp, #local_20]!
0011088c fd d3 c1 c2     mov      c29,csp
00110890 80 00 80 90     adrp     c0,0x120000
00110894 00 e8 42 c2     ldr      c0,[c0, #offset ->__dso_han
00110898 e9 03 1f aa     mov      x9,xzr
0011089c 00 00 40 c2     ldr      c0,[c0]=>->__dso_handle
001108a0 38 00 00 94     bl       <EXTERNAL>::__cxa_finalize
001108a4 fd 7b c1 22     ldp      c29,c30,[csp], #0x20
```

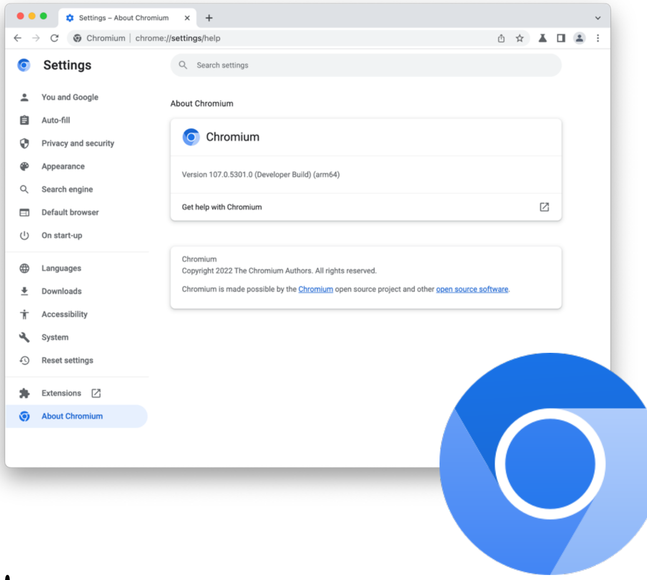
```
4 {
5   undefined *__ptr;
6   char **argv-local;
7   int a
8
9   __ptr = (undefined *)malloc(1);
10  if (__ptr == (undefined *)0x0) {
11    err((uint16)1,ZEXT816("malloc"));
```

Memory-safe OS kernel and libraries

DEMONSTRATION

Grand challenge application: Google Chromium

- Foundation for Google Chrome, Microsoft Edge, Microsoft Teams, Electron, …
  - **~27MLoC** base source code with (at least) **7** compilers, of which **6** are Just-In-Time compilers (JITs)
  - **~47MLoC** including >**760** library dependencies
  - V8, an intimidatingly real runtime for JavaScript and WASM (~2MLoC)
  - Code from diverse origins and in idiomatic C and C++
  - Vast wealth of past vulnerabilities to use in evaluation (~1 critical memory-safety vulnerability every 2 days in 2023)
  - Performance critical components, especially V8
- Chromium able to browse increasingly complex web sites
  - Roughly 18 staff months of effort so far; close to functional browsing
  - .. but .. compare to size of **100-member Chromium security team**!
  - Chromium base (27MLoC) **~0.045%** LoC change; V8 (2MLoC) **~0.8%** LoC change
- Pilot project funded by UKRI and Google



About Chromium

Chromium

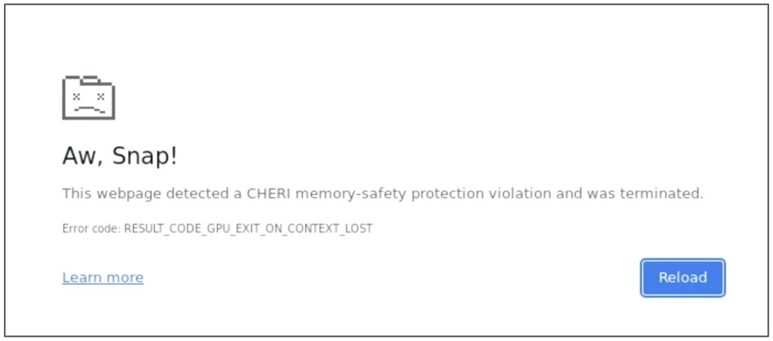
Version 107.0.5301.0 (Developer Build) (arm64)

Get help with Chromium

Chromium
Copyright 2022 The Chromium Authors. All rights reserved.

Chromium is made possible by the Chromium open source project and other open source software.

Chromium demo: CVE-2023-4863 (“BLASTPASS”)



Aw, Snap!

This webpage detected a CHERI memory-safety protection violation and was terminated.

Error code: RESULT_CODE_GPU_EXIT_ON_CONTEXT_LOST

CHERI deterministically mitigates this Chromium vulnerability without any awareness about the nature, location, or origin of the vulnerability during development.

This memory-safety vulnerability was discovered “in the wild” following targeted attacks against victims in the US using NSO Group’s Pegasus:

- **Naturally occurring vulnerability** in Google’s libwebp image library
  - Heap-memory buffer overflow exploitable for remote arbitrary code execution
  - Undiscovered for years despite fuzzing due to complexity of Huffman coding logic
- Affected Chrome, Edge, and WebKit
  - 1st-party code for Google
  - 3rd-party for Apple and Microsoft
  - Zero interaction exploitation of Apple iOS
- No prior awareness of this CVE in our work
- 0% LoC changes to webp for use on CHERI

CONCLUSION

Ease of adoption compared to high-level languages

Language	Approximate open-source LoC*	Memory safe
C	10,317,799,775	✗ → ✓ **with CHERI**
C++	2,937,552,905	✗ → ✓ **with CHERI**
Java	2,614,800,470	✓
Rust	39,538,172	✓

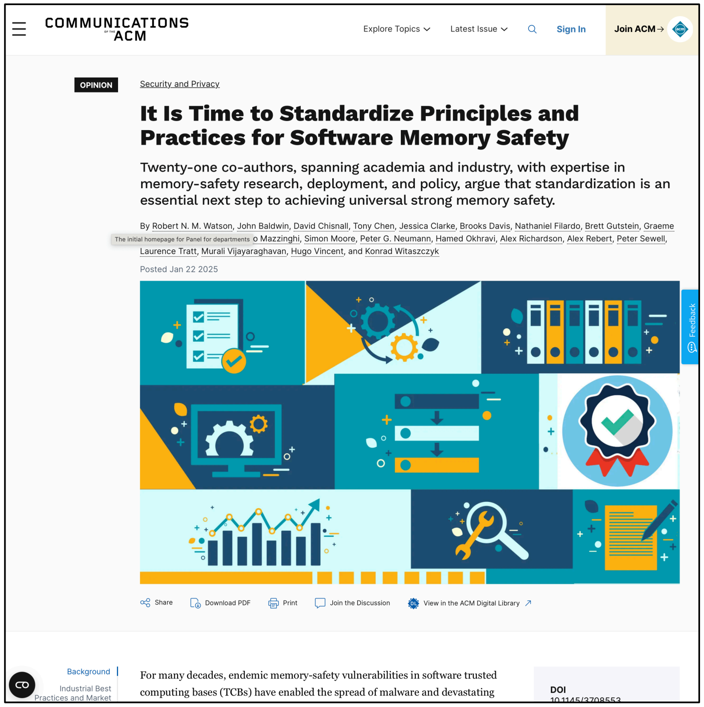
Worth pondering: In just the past 18 months, a small research team at SRI, Cambridge, and CapLtd has adapted ~150MLoC of C++ for CHERI memory safety

* Synopsys Black Duck Open Hub: https://www.openhub.net/languages - Stats taken 13 December 2023

Could we achieve practical memory safety* for multi-BLoC C/C++ software stacks within 4 years without a ground-up rewrite?

* There's a **very** long discussion to have about what "memory-safe C/C++" means, but Microsoft's practical definition of "deterministically mitigates security vulnerabilities" seems a good place to start.

It is Time to Standardize Principles and Practices for Software Memory Safety (CACM Feb 2025)



It Is Time to Standardize Principles and Practices for Software Memory Safety

Twenty-one co-authors, spanning academia and industry, with expertise in memory-safety research, deployment, and policy, argue that standardization is an essential next step to achieving universal strong memory safety.

By Robert N. M. Watson, John Baldwin, David Chisnall, Tony Chen, Jessica Clarke, Brooks Davis, Nathaniel Filardo, Brett Gutstein, Graeme The initial homepage for Panel for departments o Mazzinghi, Simon Moore, Peter G. Neumann, Hamed Okhravi, Alex Richardson, Alex Rebert, Peter Sewell, Laurence Tratt, Murali Vijayaraghavan, Hugo Vincent, and Konrad Witaszczyk

Posted Jan 22 2025

For many decades, endemic memory-safety vulnerabilities in software trusted computing bases (TCBs) have enabled the spread of malware and devastating

DOI
10.1145/3708553

- 20 coauthors including from Arm, Microsoft, and Google.
- Reflects on maturing strong memory-safety techniques for lower-level TCBs
  - Memory-safe systems languages
  - Architectural techniques such as CHERI
  - Formal methods
- The lack of a clear way to express memory-safety requirements impedes demand signals from government industry, limits policy interventions
  - E.g., no consistent understanding of ideas around **completeness** and **determinism** – e.g., "CHERI" vs "PAC and MTE"
  - Recognize and encourage current best practices while creating a tool to incentivize adoption of **strong memory-safety techniques**
- ETSI TC CYBER kicked off an effort to establish core memory-safety definitions as of August 2025 – first in-person meeting September 2025

(Draft) definitions for memory safety

Work-in-progress

- **Strong memory safety** is memory safety that **must be**:
  - **Deterministic** – Not dependent on secrets that may be leaked or brute forced
  - **Complete** – Implements referential, spatial, temporal, and control-flow memory safety
  - **Concurrency safe** – Memory safety is enforced in the presence of concurrency
- **Strong memory safety** is memory safety that **may support**:
  - **Compartmentalisation** – Retains memory safety in the presence of code compiled with an untrustworthy compiler
  - **Memory-safe sub-allocators** – Enables extensible memory allocation
  - **A memory-safety TCB** – Depends only on code that is, itself, memory safe
  - **General concurrency safety** – Strong concurrency properties for all memory accesses
- **Weak memory safety** incompletely implements required strong memory-safety properties – e.g., is probabilistic or secrets based, omits strong control-flow protections, etc.

CHERI Alliance

CHERI

axiomise
cantech
CAPABILITIES LIMITED
CHEVIN TECHNOLOGY
Codasip

Critical technologies inc.
CYNAM
CATAPULT Digital
discribe
[dstl]

FreeBSD FOUNDATION
Google
King Abdullah University of Science and Technology
LMT 光溥科技股份有限公司 LIGHT MOMENTUM TECHNOLOGY CORP.
lowRISC

National Cyber Security Centre a part of GCHQ
nquiringminds
PARVAT
RPTU
SCI Semiconductor

SRI
Prifysgol Abertawe Swansea University
TechWorks
TRUSTED COMPUTING CENTER OF EXCELLENCE
UNIVERSITY OF BIRMINGHAM

UNIVERSITY OF CAMBRIDGE
University of Glasgow
University of Kent
ZAYA

Not-for-profit membership organization that aims to:

- Support a collaborative community around CHERI from industry, academia, government
- Foster community open-source projects
  - CHERI LLVM, CHERI QEMU, CHERI-seL4, CHERI Linux, test suites, etc.
- Develop documentation, tutorials, standards, and promotional material about CHERI
- Distribute reference hardware to open-source and other developers (e.g., of Arm Morello, lowRISC Sonata boards)
- Host conferences and other events

Launched in November 2024 and membership growing as companies continue to come on board

Learning more about CHERI

- Visit **cheri-cpu.org**
- Read our article in the 2024 special issue of IEEE Security and Privacy Magazine:

  CHERI: Hardware-Enabled C/C++ Memory Protection at Scale
- See our technical reports for great detail:

  Introduction to CHERI

  CHERI C/C++ Programming Guide

  CHERI ISA Specification
- And see our research papers on everything from microarchitectural implementations of tagged memory to the implications of memory safety for the UNIX process model

Robert N.M. Watson, David Chisnall, Jessica Clarke, Brooks Davis, Nathaniel Wesley Filardo, Ben Laurie, Simon W. Moore, Peter G. Neumann, Alexander Richardson, Peter Sewell, Konrad Witaszczyk, and Jonathan Woodruff. **CHERI: Hardware-Enabled C/C++ Memory Protection at Scale**. IEEE Security & Privacy, vol. 22, no. 04, pp. 50-61, July-August 2024.

Conclusion

- New architectural primitives enable fine-grained C/C++ memory protection and scalable software compartmentalization
- Ideas portable across a range of architectures (Arm, RISC-V, …) with full-scale software stacks running on them
- Prototype Arm Morello board shipped in 2022
- Open-source Microsoft CHERIoT microcontroller released in 2023; this and other CHERI-RISC-V cores to appear in ASIC products over the next year
- Large-scale software demonstrators starting to come to fruition, with 150MLoC of C/C++ including early Chromium/V8 adaptation
- Large and active research ecosystem spanning companies that include Arm, Google, Microsoft, HP, and many others

http://www.cheri-cpu.org/