CHERI
Capability Hardware Enhanced RISC Instructions

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, 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

University of Cambridge and SRI International

Web Slide Deck – 16 January 2024

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 contracts 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"), and HR001122S0003 ("MTSS"). 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.

Approved for public release; distribution is unlimited.

This work was supported in part by the Innovate UK project 105694 ("Digital Security by Design (DSbD) Technology Platform Prototype", and Innovate UK project 10027440 ("Developing and Evaluating an Open-Source Desktop for Arm Morello").

This work was also 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"), and HR001123C0031 ("MTSS") as part of the DARPA I2O CRASH, I2O MRC, and MTO SSITH 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.

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.

CHERI 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 CPU, SoC; board announced 2019, with support from UKRI
    Shipping as of Jan 2022
  - Microsoft CHERIoT (RISC-V) Ibex core announced Sep 2022
    Open sourced in February 2023; lowRISC FPGA board announced Sep 2023
- Today's talk:
  - What is CHERI?
  - Transition efforts including Arm, Google, Microsoft, and beyond …
- http://www.cheri-cpu.org/
- Watson, et al. **An Introduction to CHERI**, UCAM-CL-TR-941, Sep. 2019.

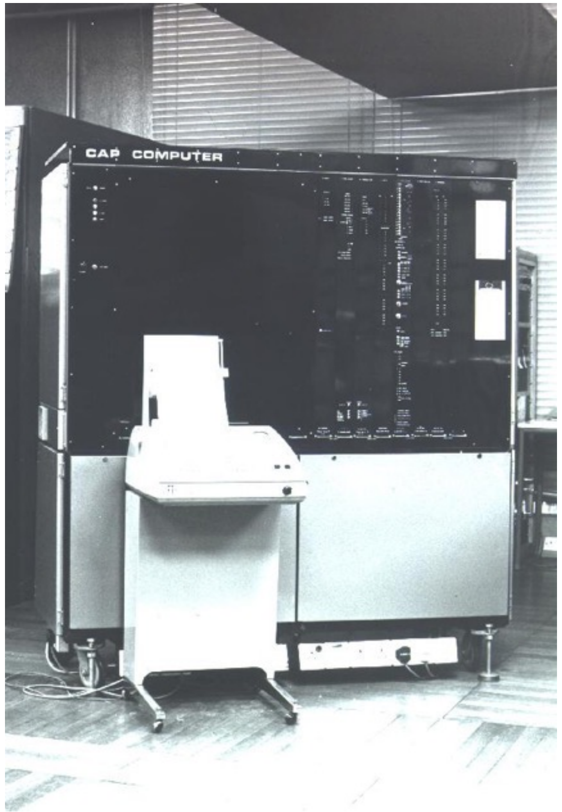
An early experimental FPGA-based CHERI tablet prototype running the CheriBSD operating system and applications, Cambridge, 2013.

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

SRI International

UNIVERSITY OF CAMBRIDGE

Capability systems

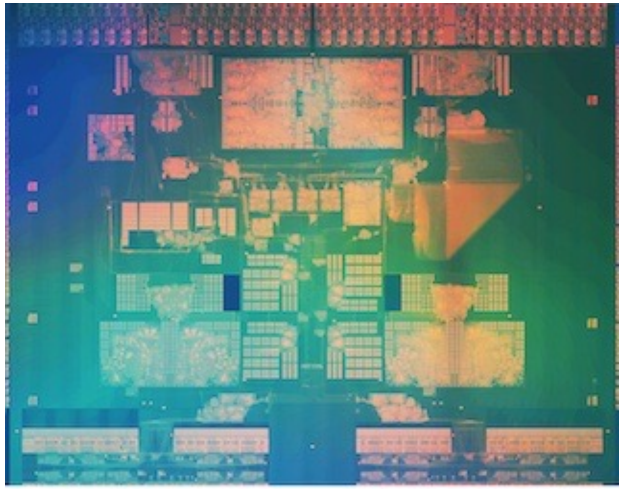


The CAP computer project ran from 1970-1977 at the University of Cambridge, led by R. Needham, M. Wilkes, and D. Wheeler.

- 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

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

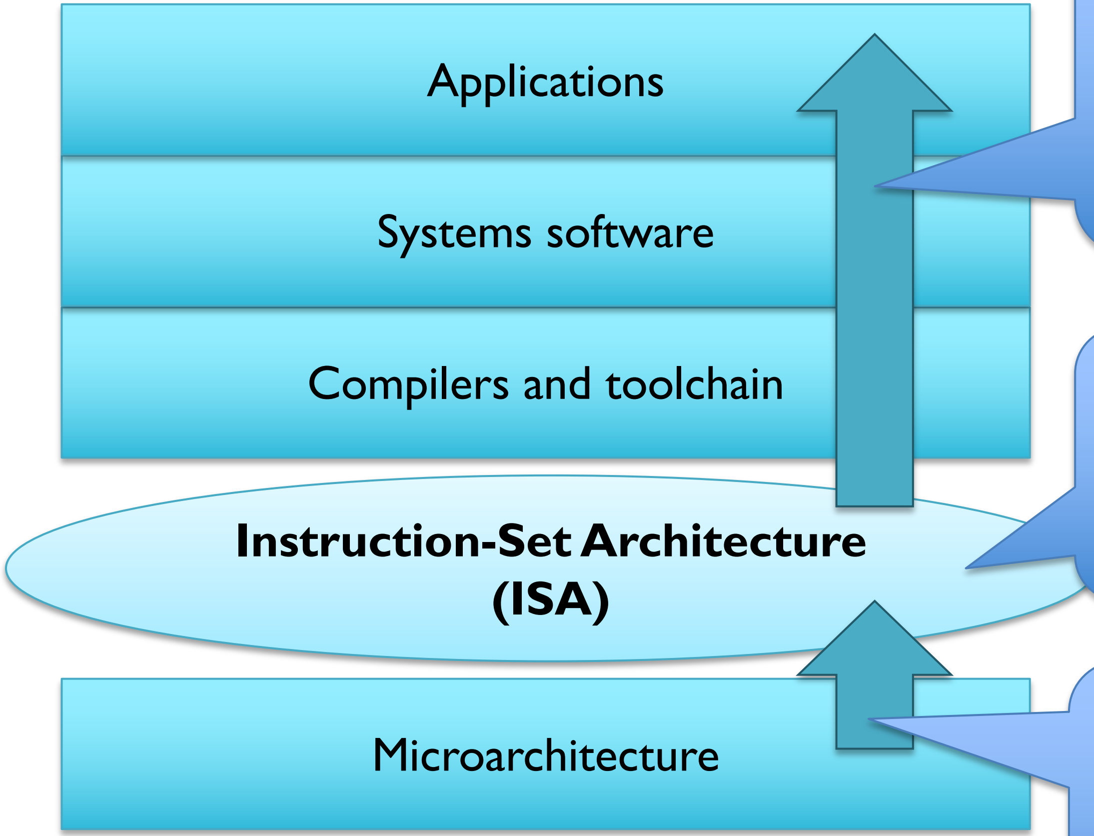


Morello chip – quad-core multi-GHz Arm processor and SoC with CHERI extensions, Arm, 2022.

SRI International

UNIVERSITY OF CAMBRIDGE

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**

SRI International

UNIVERSITY OF CAMBRIDGE

An Introduction to CHERI

Technical Report UCAM-CL-TR-941 ISSN 1476-2986

Number 941

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

An Introduction to CHERI

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

September 2019

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

- Watson, et al. **An Introduction to CHERI**, UCAM-CL-TR-941, September 2019
  - Architectural capabilities and the CHERI ISA
  - CHERI microarchitecture
  - ISA formal modeling and proof
  - Software construction with CHERI
  - Language and compiler extensions
  - OS extensions
  - Application-level adaptations

NB: Predates public announcement of Morello

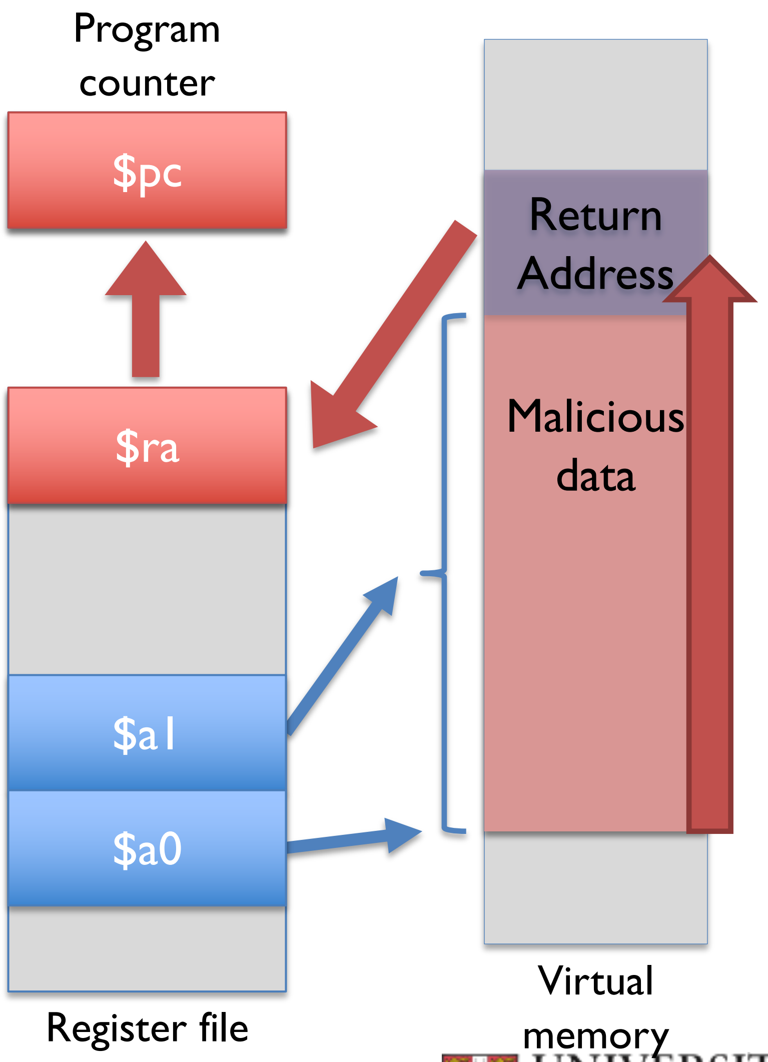
SRI International

UNIVERSITY OF CAMBRIDGE

(Lack of) architectural least privilege

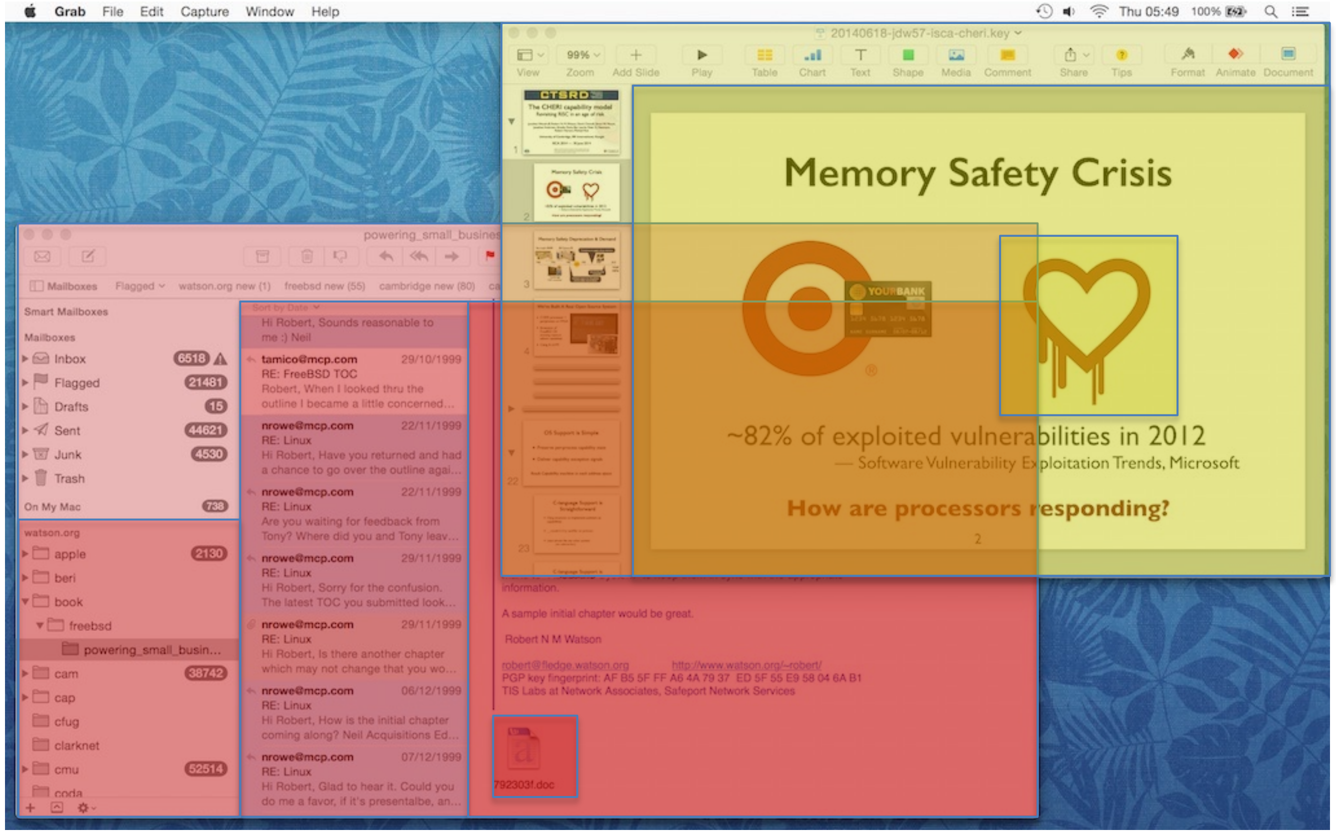
- Classical buffer-overflow attack
  1. Buggy code overruns a buffer, overwrites return address with attacker-provided value
  2. Overwritten return address is loaded and jumped to, allowing the attacker to manipulate control flow
- These privileges were not required by the C language; why allow code the ability to:
  - Write outside the target buffer?
  - Corrupt or inject a code pointer?
  - Execute data as code / re-use code?
- Limiting privilege doesn't fix bugs – but does provide **vulnerability mitigation**

➢ Memory Management Units (MMUs) do not enable **efficient, fine-grained privilege reduction**



Application-level least privilege

Software compartmentalization decomposes software into **isolated compartments** that are delegated **limited rights**

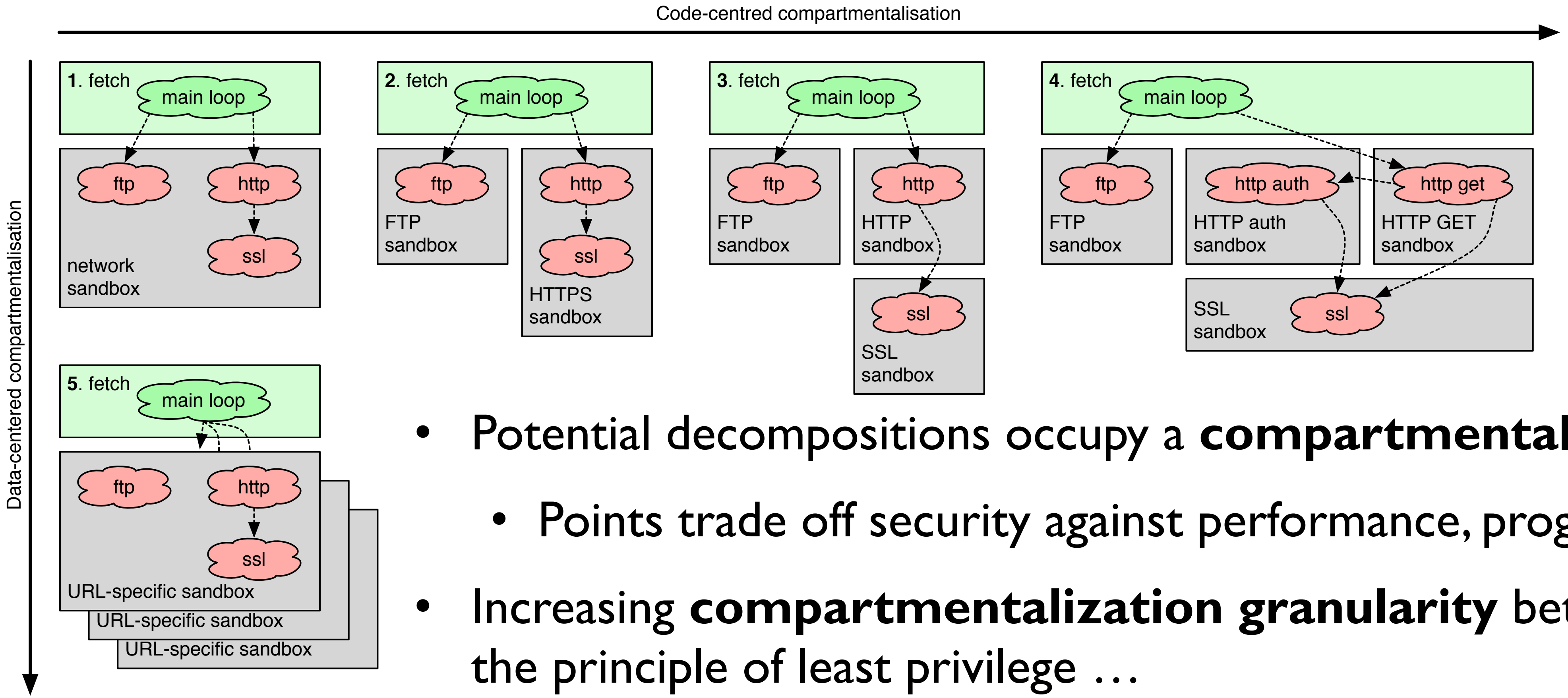


Potential compartmentalization boundaries matching reasonable user expectations for **least privilege** can be found in many user-facing apps.

E.g., a malicious email attachment should not be able to gain access to other attachments, messages, folders, accounts, or the system as a whole.

Able to mitigate not only **unknown vulnerabilities**, but also **as-yet undiscovered classes of vulnerabilities and exploits**

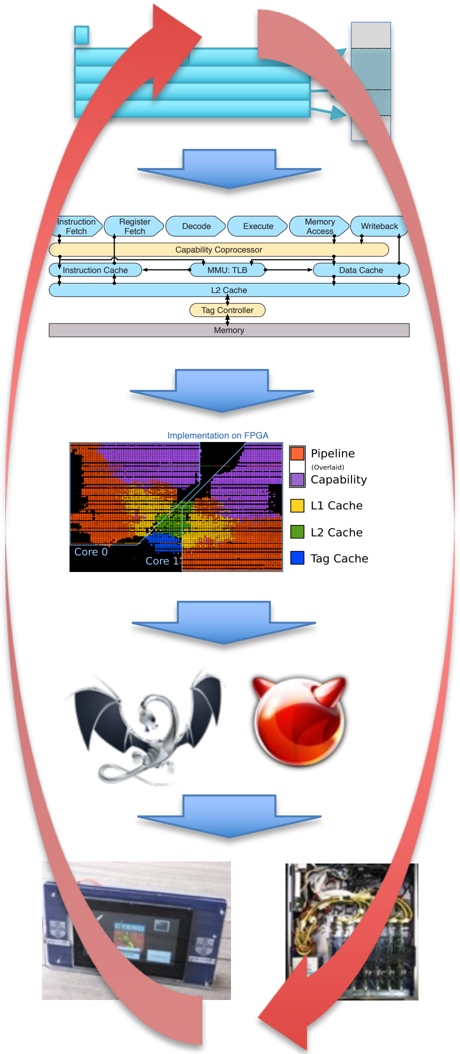
SRI International
UNIVERSITY OF CAMBRIDGE



- Potential decompositions occupy a **compartmentalization space**:
  - Points trade off security against performance, program complexity
- Increasing **compartmentalization granularity** better approximates the principle of least privilege …
- … but **MMU-based architectures** do not scale to many processes:
  - Poor spatial protection granularity
  - Limited simultaneous-process scalability
  - Multi-address-space programming model

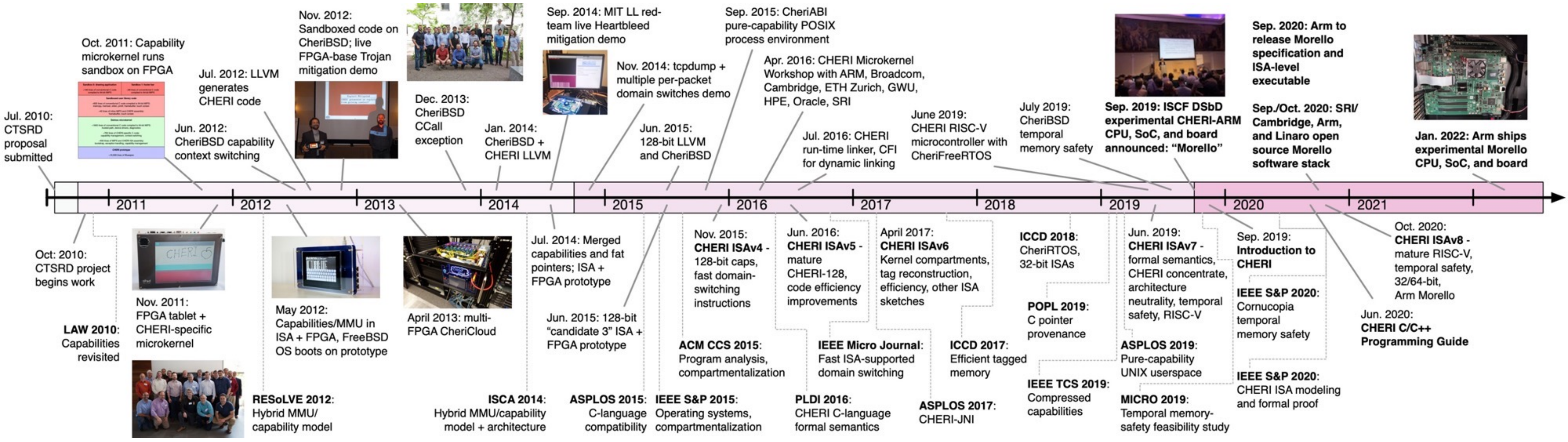
HARDWARE-SOFTWARE CO-DESIGN FOR CHERI

Hardware-software-semantics co-design



- University of Cambridge and SRI International from 2010 supported by DARPA
- Architectural mitigation for C/C++ TCB vulnerabilities
  - Tagged memory, new hardware capability data type
  - Model hybridizes cleanly with contemporary hardware and software designs
  - New hardware enables incremental software deployment
- Hardware-software-semantics co-design + concrete prototyping:
  - CHERI abstract protection model; concrete ISA instantiations in 64-bit MIPS, 32/64-bit RISC-V (+ Microsoft CHERIoT), 64-bit Armv8-a (Arm Morello)
  - Formal ISA models, Qemu-CHERI, and multiple FPGA prototypes
  - Formal proofs that ISA security properties are met, automatic testing
  - CHERI Clang/LLVM/LLD, CheriBSD, C/C++-language applications
  - Repeated iteration to improve {performance, security, compatibility, ..}

CHERI research and development timeline



Years 1-2: Research platform, prototype architecture

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

Years 4-7: Efficiency, CheriABI/C/C++/linker, ARMv8-A

Years 8-12: RISC-V, temporal safety, proof, Arm Morello, Microsoft CHERI Ibex

CHERI ISA refinement over 13 years

Technical Report

UCAM-CL-TR-987
ISSN 1476-2986

Number 987

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

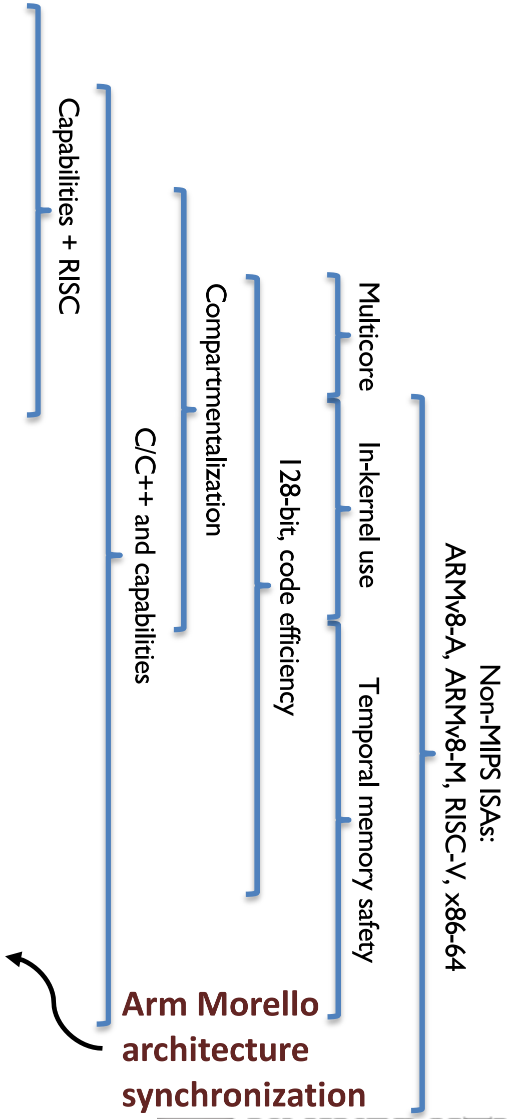
Capability Hardware
Enhanced RISC Instructions:
CHERI Instruction-Set Architecture
(Version 9)

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

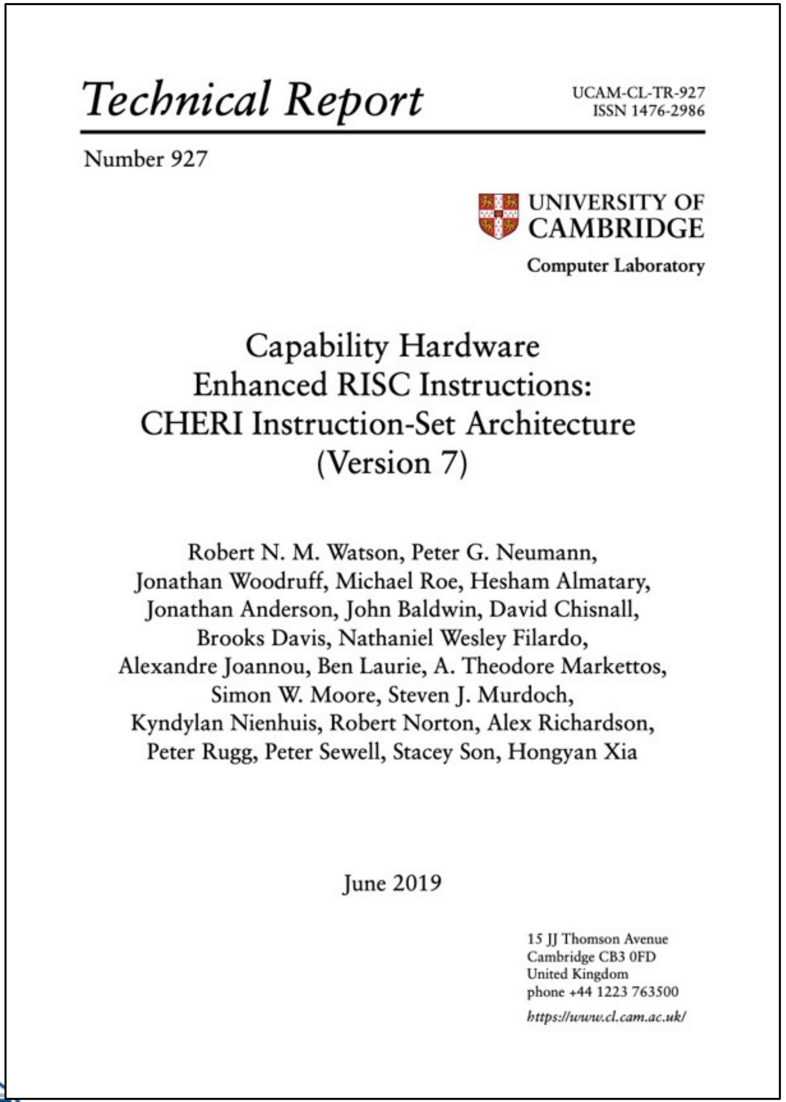
15 JJ Thomson Avenue
Cambridge CB3 0FD
United Kingdom
phone +44 1223 763500
https://www.cl.cam.ac.uk/

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 ISAv7 – June 2019



Technical Report

UCAM-CL-TR-927
ISSN 1476-2986

Number 927

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Capability Hardware
Enhanced RISC Instructions:
CHERI Instruction-Set Architecture
(Version 7)

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

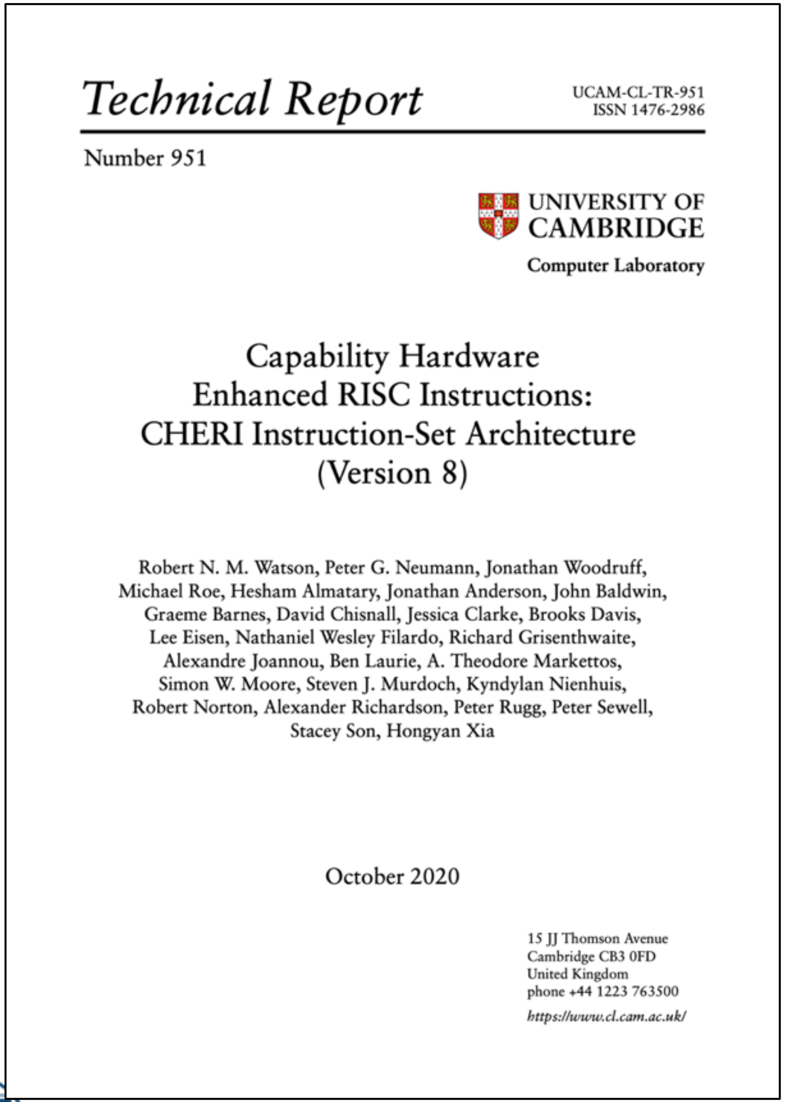
June 2019

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

- Key features:
  - Architecture-neutral CHERI model
  - Elaborated CHERI-RISC-V ISA
  - CHERI Concentrate capability compression (**IEEE TC 2019**)
  - Side-channel resistance features
  - Improved C-language compatibility, dynamic linkage, performance optimizations (**ASPLOS 2019**)
  - Experimental features including 64-bit capabilities for 32-bit architectures (**ICCD 2018**), temporal safety (**IEEE Micro 2019, IEEE SSP 2020**)
  - All instruction pseudocode derived from Sail formal models, formally proven properties (**IEEE SSP 2020**)

UNIVERSITY OF CAMBRIDGE

CHERI ISAv8 (October 2020)



Technical Report UCAM-CL-TR-951 ISSN 1476-2986

Number 951

UNIVERSITY OF CAMBRIDGE

Computer Laboratory

Capability Hardware
Enhanced RISC Instructions:
CHERI Instruction-Set Architecture
(Version 8)

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, 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

October 2020

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

- Key changes
  - Capability compression is now part of the abstract protection model
  - Both 32-bit and 64-bit architectural address sizes are supported
  - Various experimental features are now mature: Sentry capabilities, CHERI-RISC-V
  - New MMU temporal memory-safety mechanisms based on load-side barrier model (**ASPLOS 2024**)
  - CHERI microarchitecture chapter
- **Synchronized with Arm Morello (IEEE MICRO Journal 2023)**

UNIVERSITY OF CAMBRIDGE

CHERI ISAv9 (September 2023)

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/

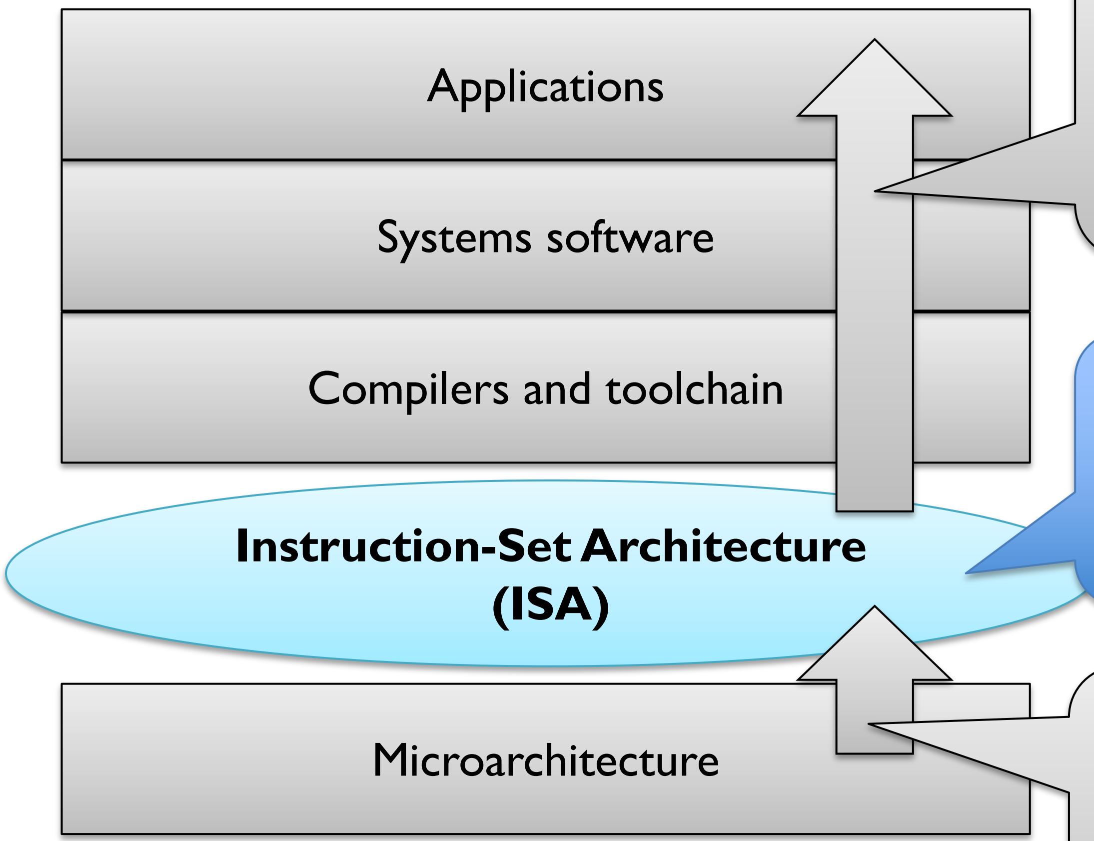
- Most recent specification version (released about every two years)
- Key changes
  - CHERI-RISC-V is now the reference architecture
  - Numerous CHERI-RISC-V improvements for standardization
  - CHERI-MIPS removed
  - CHERI-x86 better elaborated
  - Complete shift to tag stripping from exception throwing for non-monotonic capability operations

CHERI: From research to product

- Starting in 2010, hardware-software co-design using FPGAs, open-source software, created a CHERI-MIPS CPU + software stack at Cambridge and SRI
- Arm collaboration from 2014, supported by DARPA; Arm Morello CPU, SoC; board announced 2019, with support from InnovateUK; Shipped in Jan 2022
  - High-performance 2.5GHz, multicore, out-of-order prototype CPU design
- Microsoft CHERIoT RISC-V CPU open sourced Feb 2023
  - 3-stage pipeline for small embedded / IoT / root-of-trust; based on Ibex
  - lowRISC FPGA board for CHERIoT announced Sep 2023; ship date in 2024
  - SCI SoC using CHERIoT announced Nov 2023; ship date in 2024
- Codasip CHERI RISC-V CPU announced in Nov 2023; ship date in 2024
  - 7-stage in-order processor line targeted at high-end embedded

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**

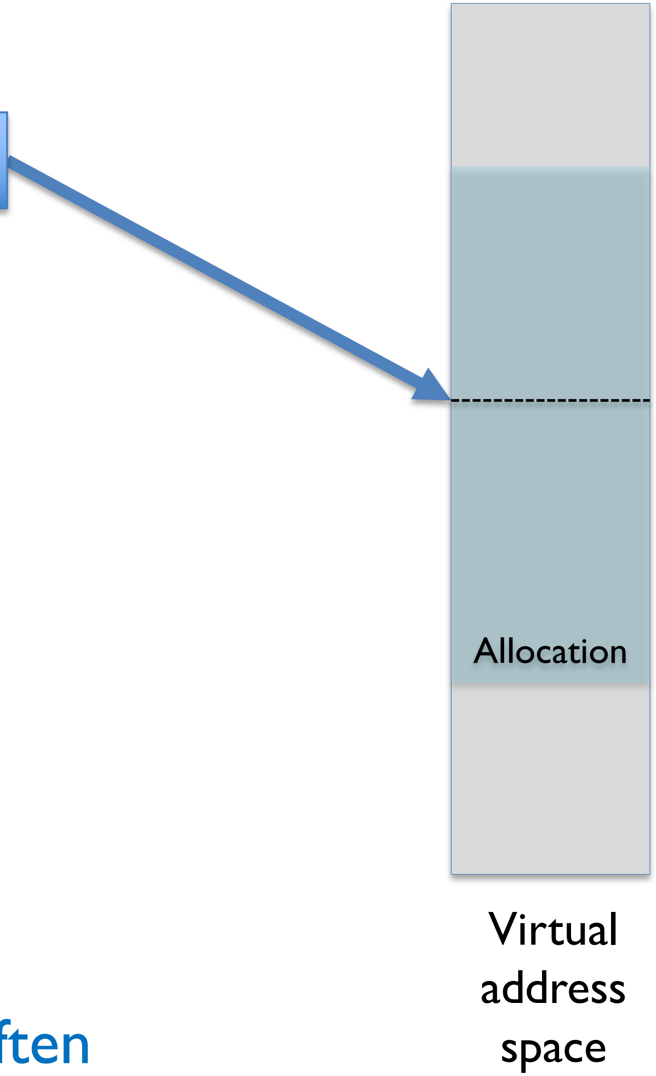
CHERI capabilities are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

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**

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, …)

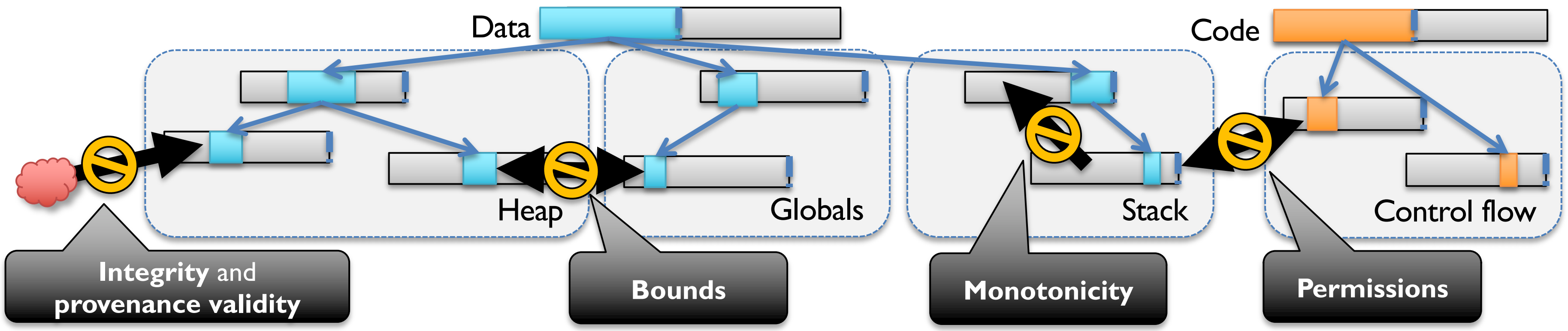
Pointers today



- Implemented as **integer virtual addresses (VAs)**
- (Usually) point into **allocations**, **mappings**
  - **Derived** from other pointers via integer arithmetic
  - **Dereferenced** via jump, load, store
- **No integrity protection** – can be injected/corrupted
- **Arithmetic errors** – out-of-bounds leaks/overwrites
- **Inappropriate use** – executable data, format strings

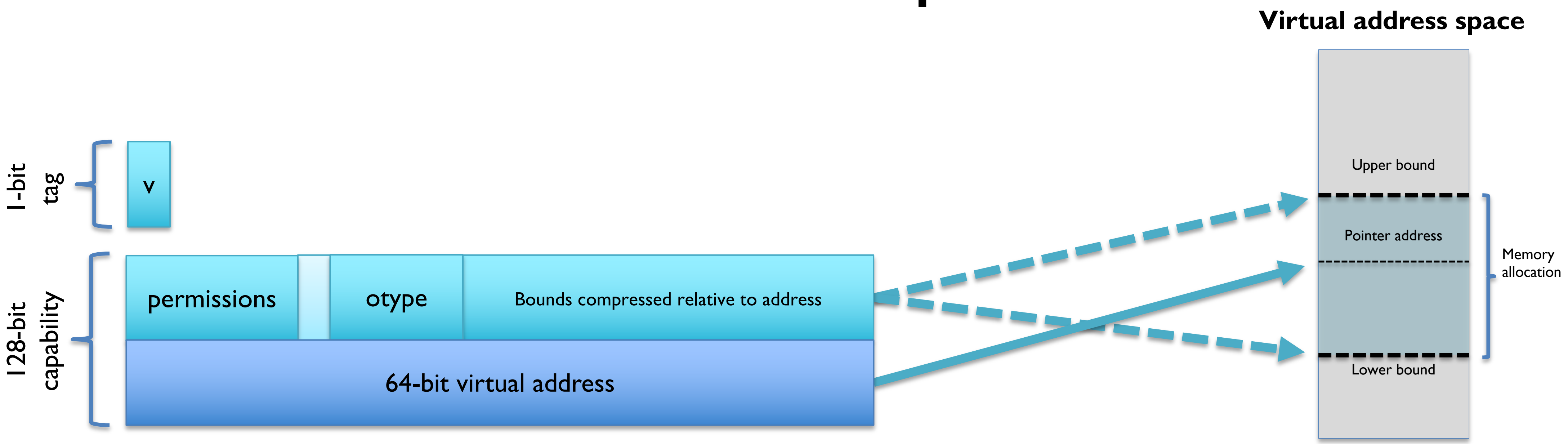
➢ Attacks on data and code pointers are highly effective, often achieving **arbitrary code execution**

CHERI enforces protection semantics for pointers



- **Integrity** and **provenance validity** ensure that valid pointers are derived from other valid pointers via valid transformations; **invalid pointers cannot be used**
  - Valid pointers, once removed, cannot be reintroduced solely unless rederived from other valid pointers
  - E.g., Received network data cannot be interpreted as a code/data pointer – even previously leaked pointers
- **Bounds** prevent pointers from being manipulated to access the wrong object
  - Bounds can be minimized by software – e.g., stack allocator, heap allocator, linker
- **Monotonicity** prevents pointer privilege escalation – e.g., broadening bounds
- **Permissions** limit unintended use of pointers; e.g., W^X for pointers
- These primitives not only allow us to implement **strong spatial and temporal memory protection**, but also higher-level policies such as **scalable software compartmentalization**

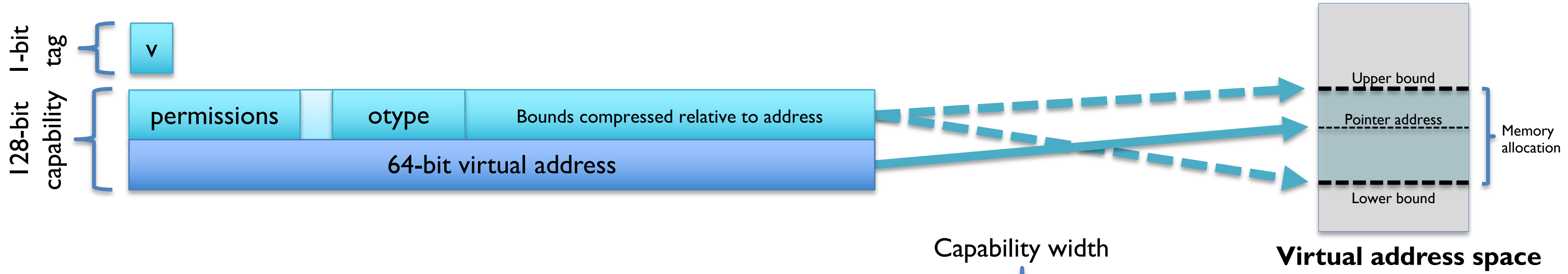
CHERI 128-bit capabilities

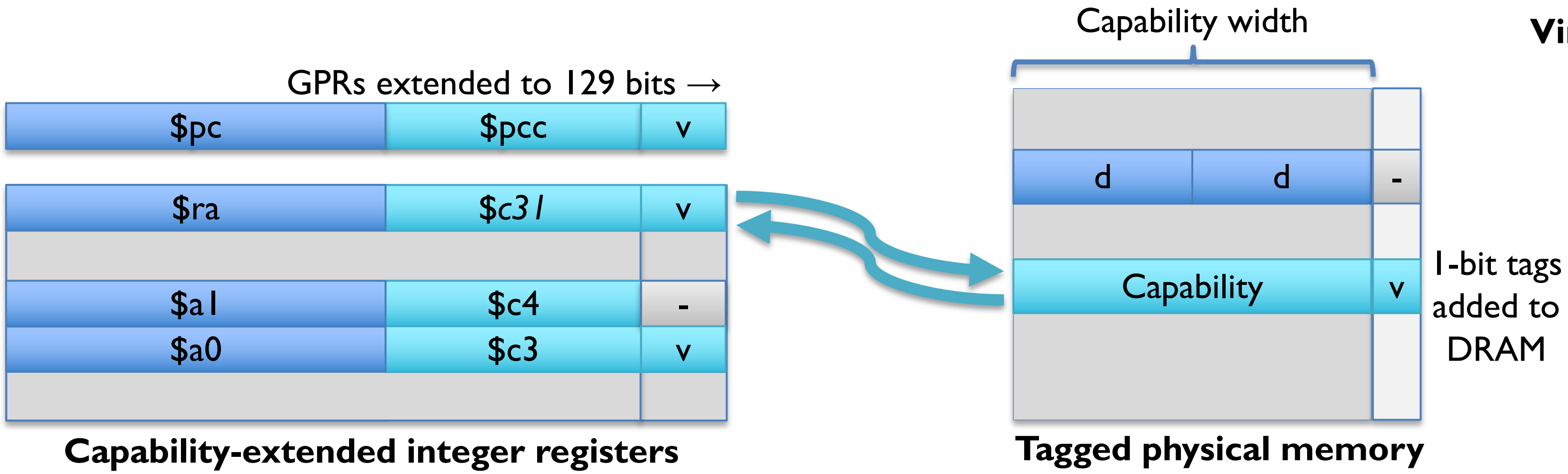


- **Capabilities** extend **integer memory addresses**
- **Metadata** (bounds, permissions, …) control how they may be used
- **Guarded manipulation** controls how capabilities may be manipulated; e.g., **provenance validity** and **monotonicity**
- **Tags** protect capability integrity/derivation in registers + memory

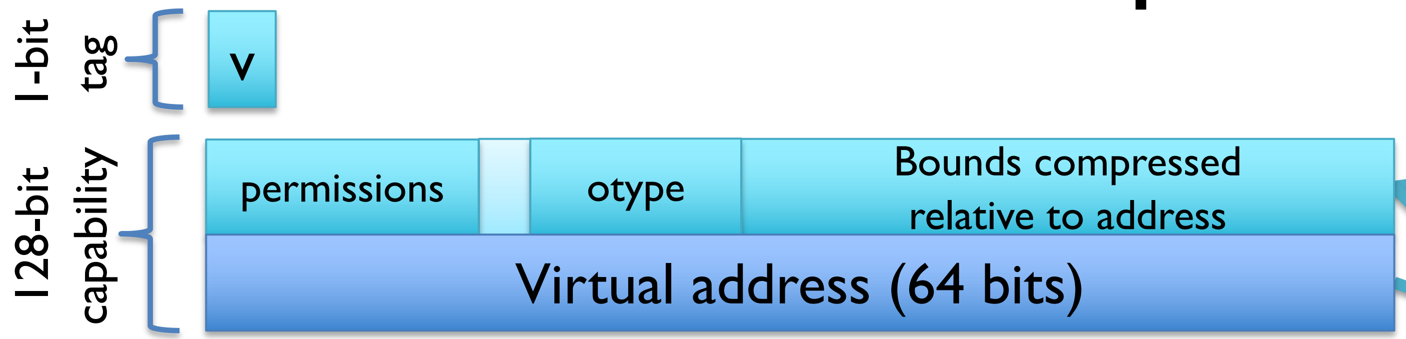
CHERI 128-bit capabilities

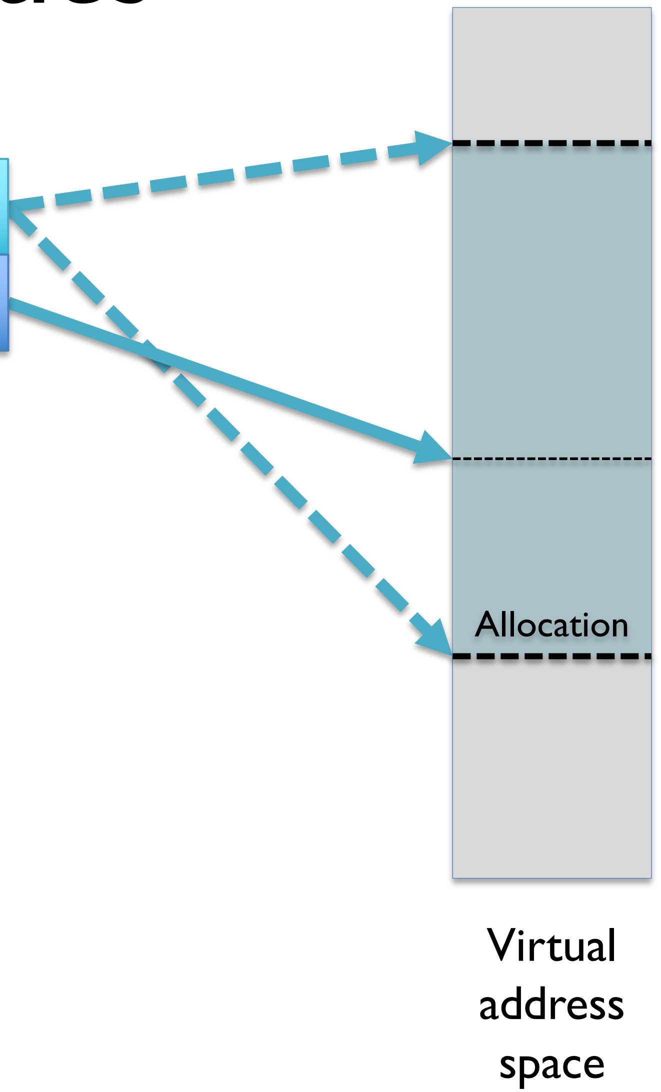
- **CHERI capabilities** are a new architectural data type extending integer addresses
- **Capability metadata** (bounds, permissions, …) control how a capability may be used
- **Capability tags** protect the integrity + safe derivation of capabilities in registers and memory





CHERI 128-bit capabilities



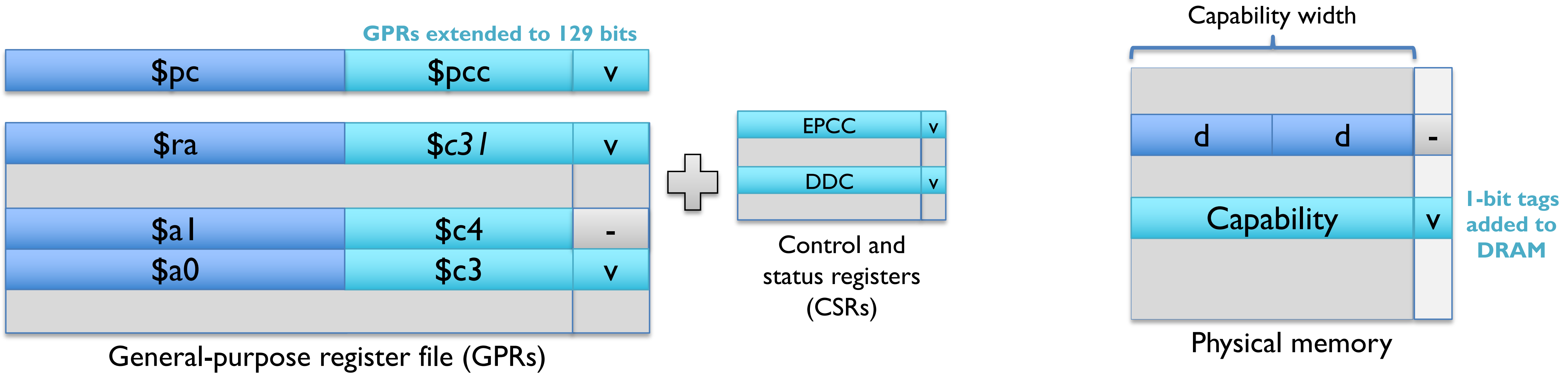


CHERI capabilities extend pointers with:

- **Tags** protect capabilities in registers and memory
  - Dereferencing an untagged capability throws an exception
  - In-memory overwrite automatically clears capability tag
- **Bounds** limit range of address space accessible via pointer
  - Floating-point compressed 64-bit lower and upper bounds
  - Strengthens larger allocation alignment requirements
  - Out-of-bounds pointer support essential to C-language compatibility
- **Permissions** limit operations – e.g., load, store, fetch
- **Sealing**: **immutable**, **non-dereferenceable capabilities** – used for non-monotonic transitions

Merged capability register file + tagged memory

(as found in Morello and CHERI-RISC-V; MIPS used a split register file)



- **64-bit general-purpose registers (GPRs)** are extended with **64 bits of metadata** and a **1-bit validity tag**
- **Program counter (PC)** is extended to be the **program-counter capability ($PCC)**
- **Default data capability ($DDC)** constrains legacy integer-relative ISA load and store instructions
- **Tagged memory** protects capability-sized and -aligned words in DRAM by adding a **1-bit validity tag**
- **Various system mechanisms** are extended (e.g., capability-instruction enable control register, new TLB/PTE permission bits, exception code extensions, saved exception stack pointers and vectors become capabilities, etc.)

CHERI-RISC-V formal ISA model

- CHERI RISC-V ISA model extends RISC-V formal ISA specification, in Sail
- Sail RISC-V ISA specification developed by UCam + SRI
  - Selected as official RISC-V spec by the Foundation
  - Sail is a custom first-order imperative language for expressing ISA specifications, usable by engineers but with static type checking of bitvector lengths etc.
  - The Sail spec is inlined in versions of the unprivileged and privileged RISC-V manuals
  - Sail auto-generates a C emulator, theorem-prover definitions, and SMT definitions
  - Machinery for configuring model WRT YAML from compliance group
  - Readable, precise definition of ISA behavior, usable as test oracle for testing hardware against and for software bring-up, and providing prover definitions if you want more rigorous reasoning
- Paper on earlier CHERI-MIPS L3 modelling and proof work at IEEE SSP 2020
- Most recently completed monotonicity proofs for the Arm Morello architecture

ISA formal modelling and verification

Rigorous engineering for hardware security:
Formal modelling and pro
and implement

Kyndylan Nienhuis*, Alexandre Joannou*, Thomas Bauer
Matthew Naylor*, Robert M. Norton*, Simon W. Moore*, P
and Peter S
*University of Cambridge †ARM Limited ‡

IEEE SSP 2020

ESOP 2022

Verified Security for the Morello
Capability-enhanced Prototype Arm Architecture

Thomas Bauereiss[1], Brian Campbell[2], Thomas Sewell[1],
Alasdair Armstrong[1], Lawrence Esswood[1], Ian Stark[2], Graeme Barnes[3],
Robert N. M. Watson[1], and Peter Sewell[1]

[1] University of Cambridge, Cambridge, UK
first.last@cl.cam.ac.uk

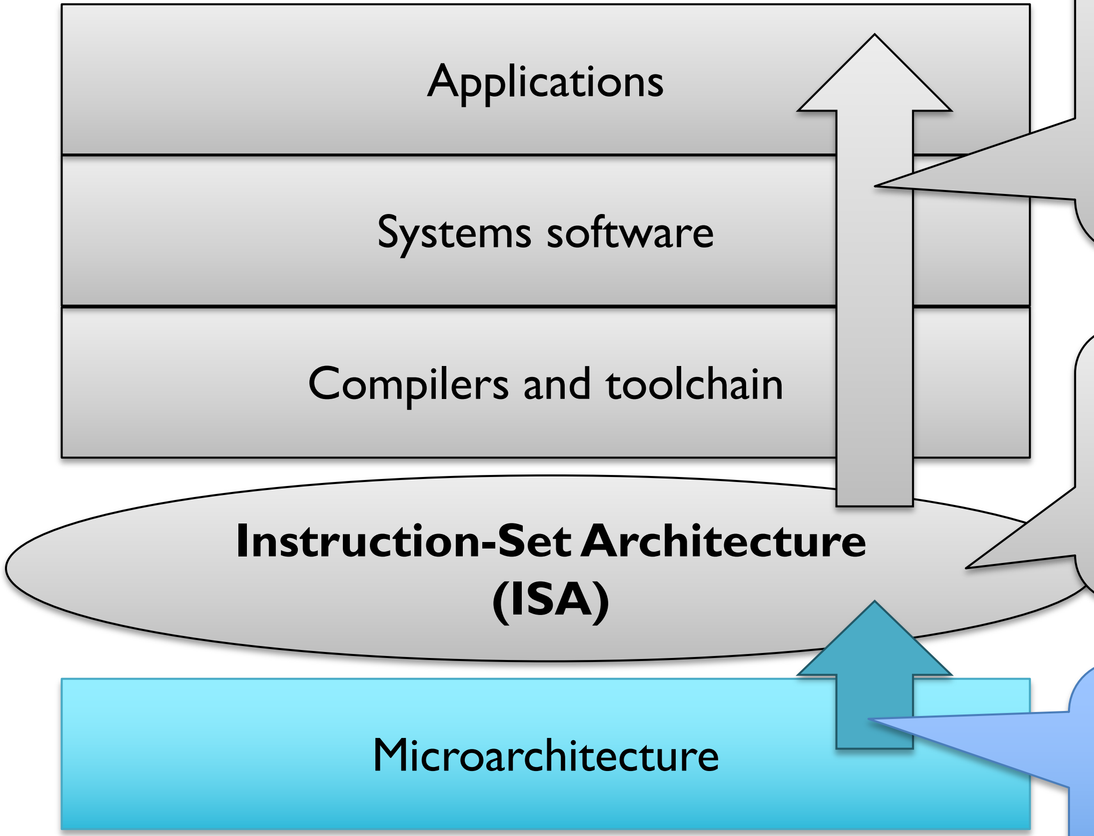
- Formal ISA models CHERI-MIPS, CHERI-RISC-V, and Morello
- Formal proof of compartmentalization for CHERI-MIPS, Morello

SRI International

UNIVERSITY OF CAMBRIDGE

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**

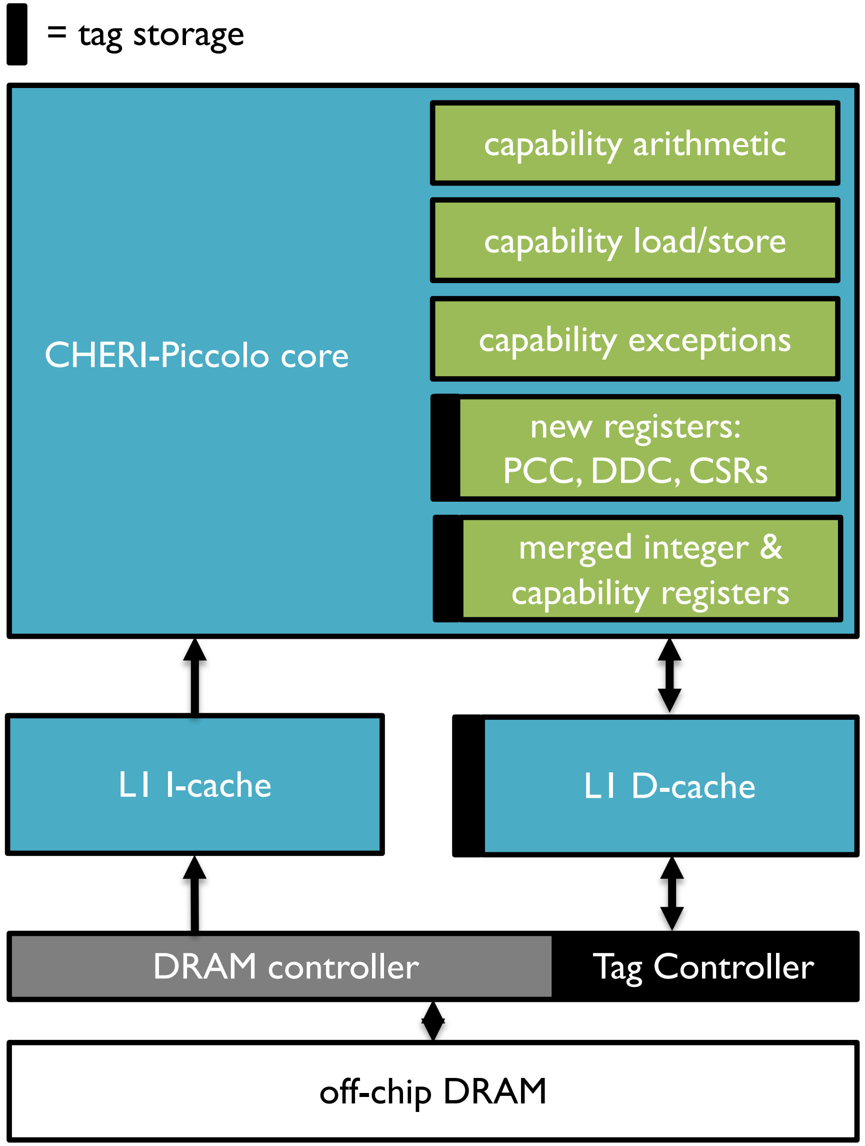
CHERI hardware research prototypes

- Original research based on our home-grown pipelined BERI MIPS core (CHERI-MIPS)
- We have transitioned our CHERI research to extended versions of open-source off-the-shelf BSV RISC-V cores (CHERI-RISC-V)
  - CHERI-Piccolo 3-stage pipeline, 32-bit, no MMU
  - CHERI-Flute 5-stage pipeline, 32- or 64-bit, MMU
  - CHERI-Toooba Superscalar, 64-bit, MMU

Our primary research processor platform

- Novel microarchitectural contributions include **capability compression model**, **tagged memory implementation techniques**
- All of our CPU designs are open source
- We also provide a QEMU full-system and userlevel simulators for CHERI-RISC-V

Example microarchitecture: CHERI-Piccolo microcontroller



Changes to the Piccolo core (RISC-V 3-stage pipeline):

- capability arithmetic
- capability load/store operations with bounds checking
- extended exception model
- PC becomes a capability (PCC)
- default data capability (DDC)
- new control/status registers
- merged integer & capability register file

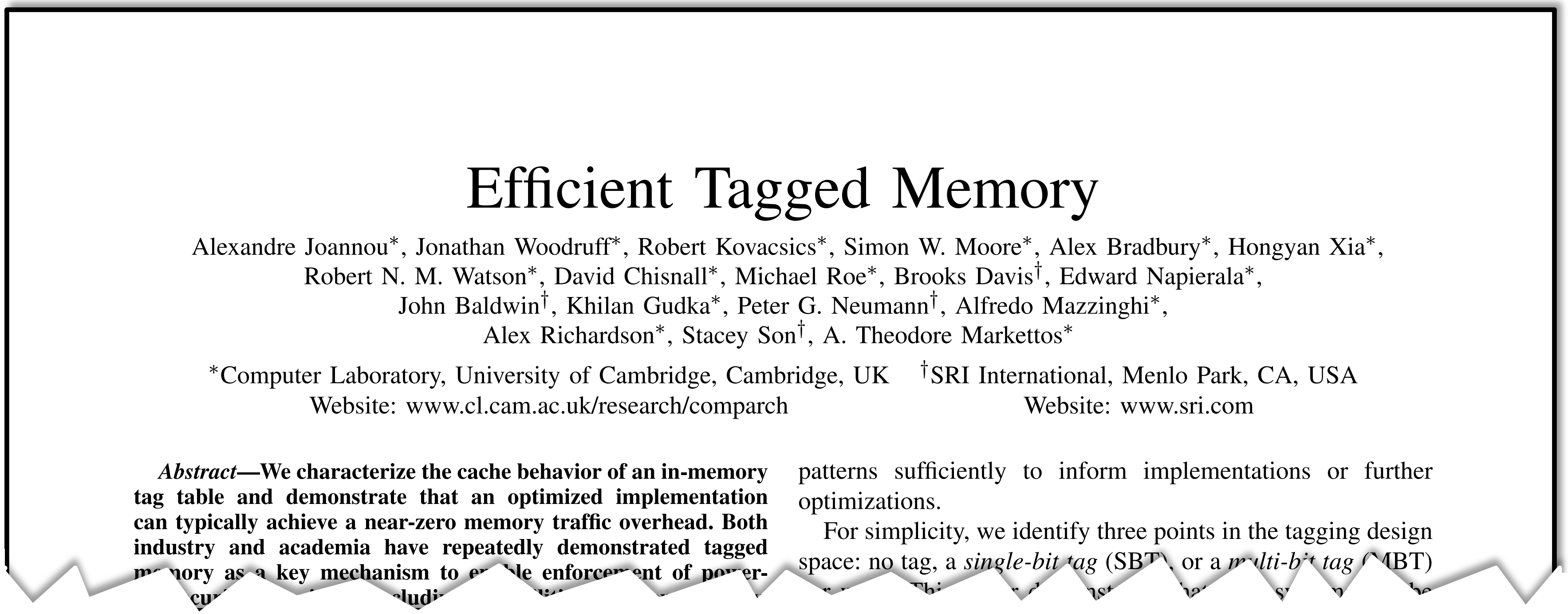
Memory subsystem:

- AXI user-field added to transport tag bits & data width doubled
- caches extended to include tags

DRAM changes:

- New tag controller uses a hierarchical tag table to efficiently store tag bits backed by top of DRAM

Microarchitectural tag storage for off-the-shelf DRAM



Efficient Tagged Memory

Alexandre Joannou*, Jonathan Woodruff*, Robert Kovacsics*, Simon W. Moore*, Alex Bradbury*, Hongyan Xia*,
Robert N. M. Watson*, David Chisnall*, Michael Roe*, Brooks Davis†, Edward Napierala*,
John Baldwin†, Khilan Gudka*, Peter G. Neumann†, Alfredo Mazzinghi*,
Alex Richardson*, Stacey Son†, A. Theodore Markettos*

*Computer Laboratory, University of Cambridge, Cambridge, UK
Website: www.cl.cam.ac.uk/research/comparch

†SRI International, Menlo Park, CA, USA
Website: www.sri.com

***Abstract*—We characterize the cache behavior of an in-memory tag table and demonstrate that an optimized implementation can typically achieve a near-zero memory traffic overhead. Both industry and academia have repeatedly demonstrated tagged**

patterns sufficiently to inform implementations or further optimizations.

For simplicity, we identify three points in the tagging design space: no tag, a *single-bit tag* (SBT), or a *multi-bit tag* (MBT)

- Published in the IEEE International Conference on Computer Design (ICCD) 2017
- Shift from flat to hierarchal tag table to hold tags in DRAM
  - Exploit inconsistent density of tags in physical memory
  - Reduces DRAM access overhead for a variety of workloads

SRI International

UNIVERSITY OF CAMBRIDGE

Compressing capability bounds

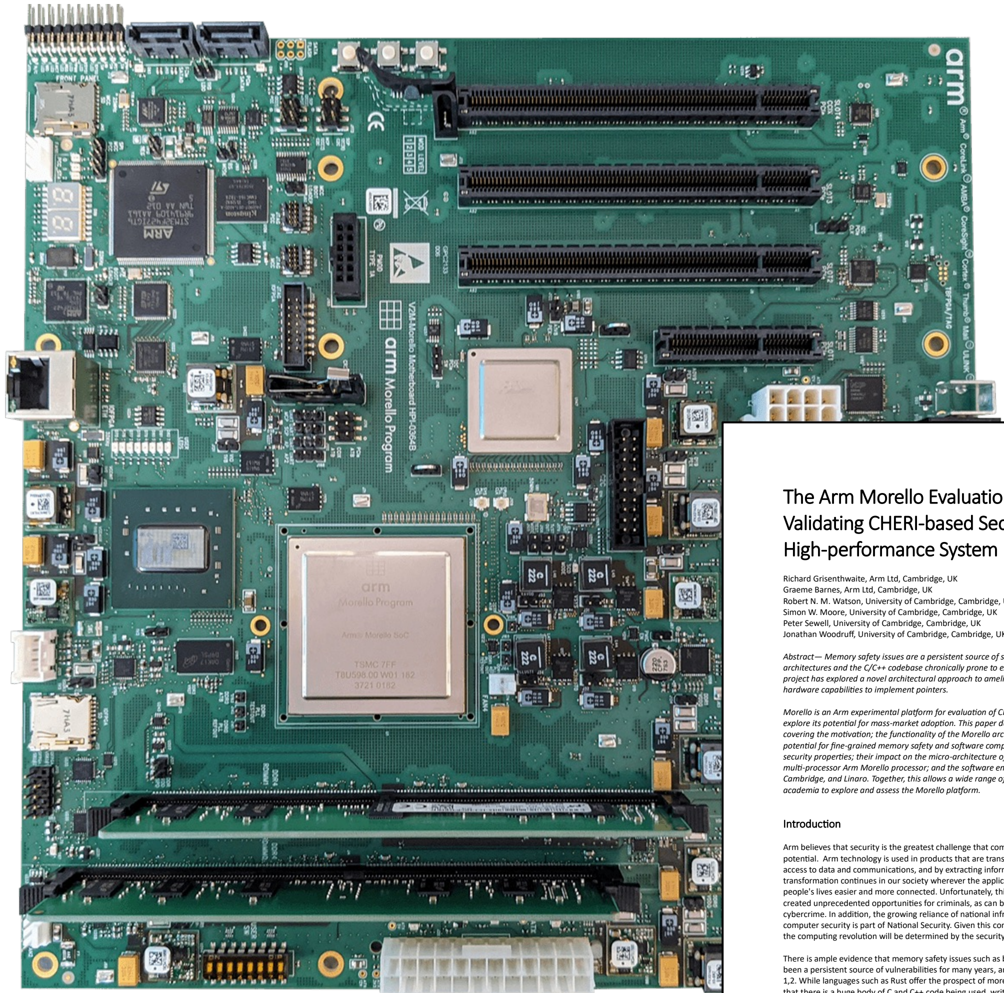
CHERI Concentrate: Practical Compressed Capabilities

Jonathan Woodruff, Alexandre Joannou, Hongyan Xia, Anthony Fox, Robert Norton, Thomas Bauereiss, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore Markettos, Michael Roe, Peter G. Neumann, Robert N. M. Watson, Simon W. Moore

Abstract—We present CHERI Concentrate, a new fat-pointer compression scheme applied to CHERI, the most developed capability-pointer system at present. Capability fat pointers are a primary candidate to enforce fine-grained and non-bypassable security properties in future computer systems, although increased pointer size can severely affect performance. Thus, several proposals for capability compression have been suggested elsewhere that do not support legacy instruction sets, ignore features critical to the existing software base, and also introduce design inefficiencies to RISC-style processor pipelines. CHERI Concentrate improves on the state-of-the-art region-encoding efficiency, solves important pipeline problems, and eases semantic restrictions of

- Published in IEEE Transactions on Computers, April 2019
- Efficient compressed capabilities for 32-bit and 64-bit processors
  - Reduces size of capabilities from 4x machine word size to 2x
  - Large reduction in cache overheads
  - Efficiently fits into a RISC pipeline with negligible impact on clock frequency
  - Maintains all security and software compatibility properties

Arm Morello (2022)



- $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

The Arm Morello Evaluation Platform-Validating CHERI-based Security in a High-performance System

Richard Grisenthwaite, Arm Ltd, Cambridge, UK
Graeme Barnes, Arm Ltd, Cambridge, UK
Robert N. M. Watson, University of Cambridge, Cambridge, UK
Simon W. Moore, University of Cambridge, Cambridge, UK
Peter Sewell, University of Cambridge, Cambridge, UK
Jonathan Woodruff, University of Cambridge, Cambridge, UK

Abstract— Memory safety issues are a persistent source of security vulnerabilities, with conventional architectures and the C/C++ codebase chronically prone to exploitable errors. The 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 paper describes the Morello Evaluation Platform; covering the motivation; the functionality of the Morello architectural hardware extensions, their potential for fine-grained memory safety and software compartmentalization; their formally proven security properties; their impact on the micro-architecture of the high-performance out-of-order multi-processor Arm Morello processor; and the software enablement program by Arm, 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.

Introduction

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 can be seen in the ever-growing cost of cybercrime. In addition, the growing reliance of national infrastructure on technology means that computer security is part of National Security. Given this context, 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. While 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 the 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.

IEEE Micro Journal 2023

SRI International

UNIVERSITY OF CAMBRIDGE

Microsoft CHERIoT core (2023)

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.

[*]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

ACM ISBN 979-8-4007-0329-4/23/10.
https://doi.org/10.1145/3613424.3614266

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

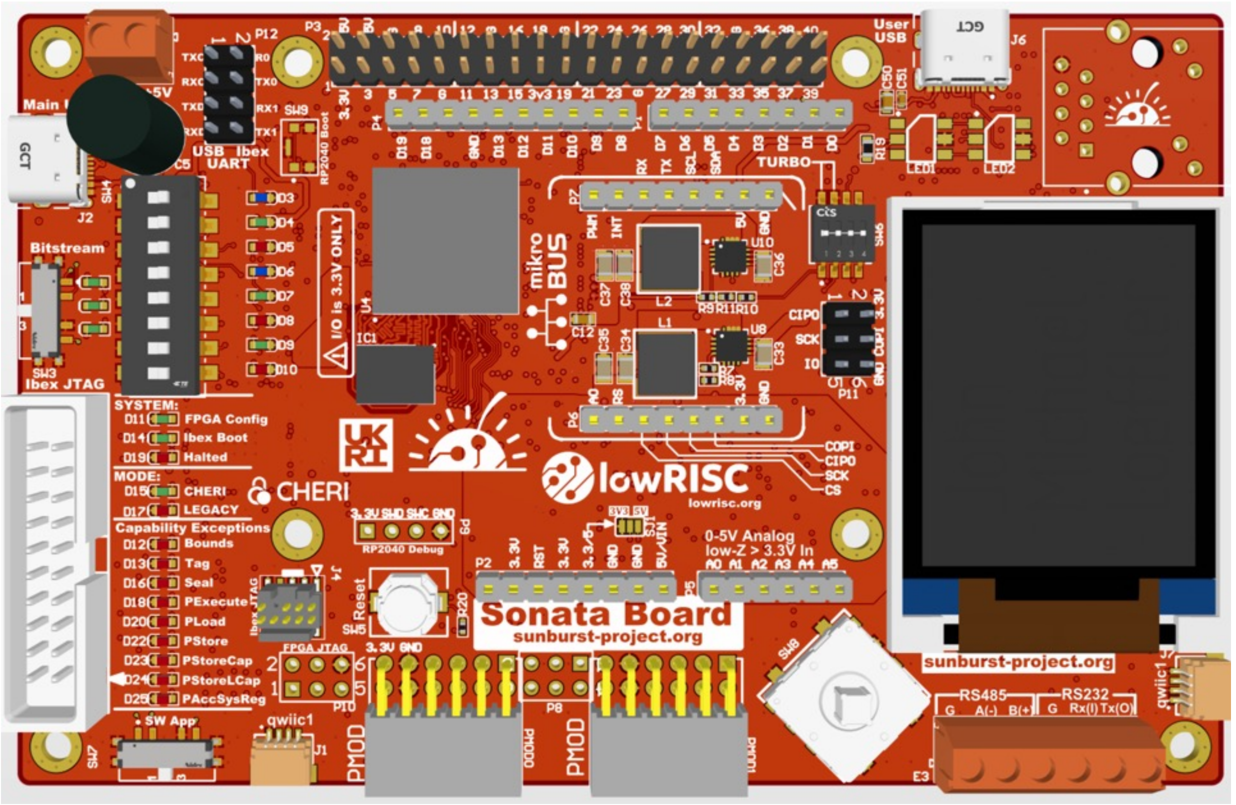
1 INTRODUCTION

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

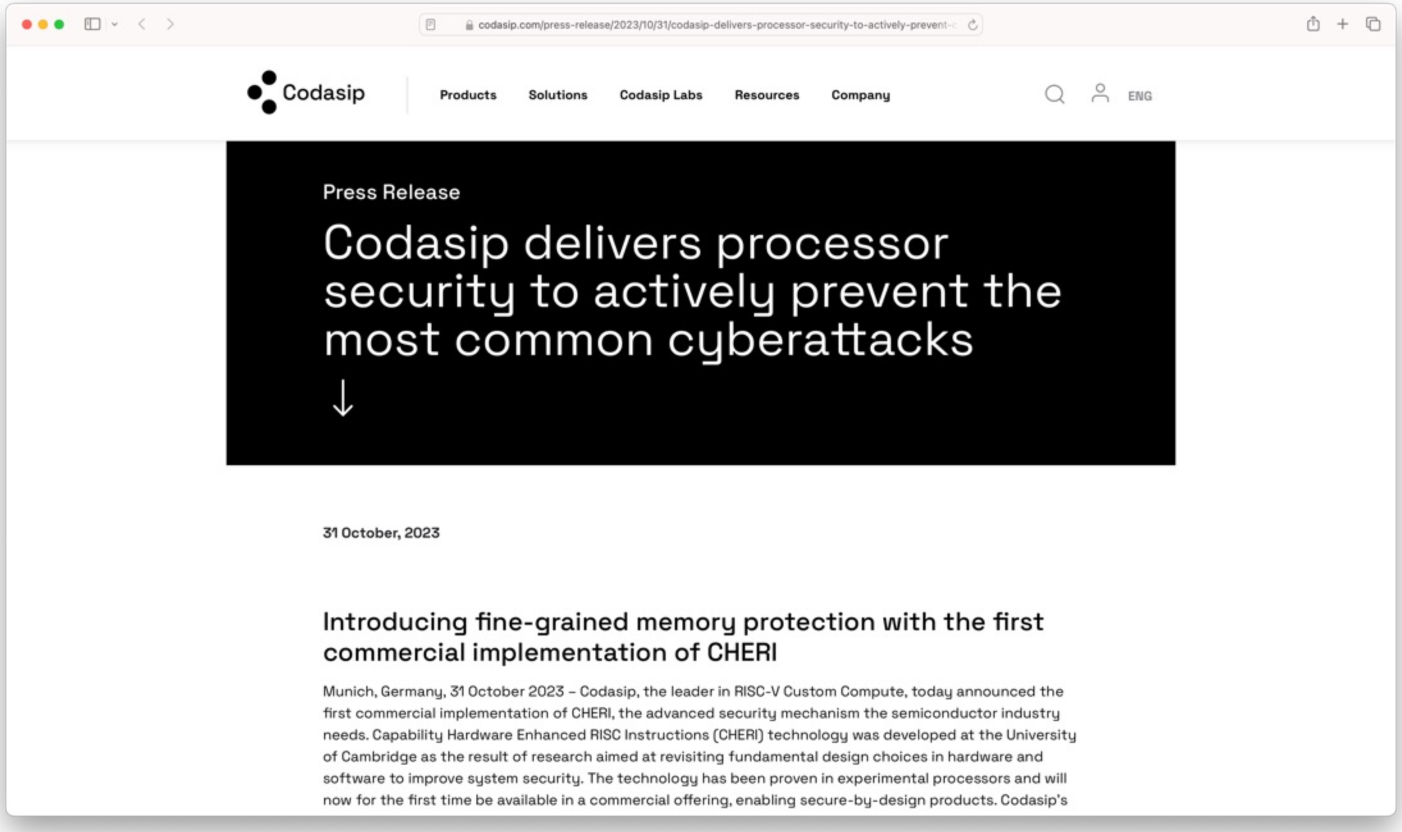
- Production CHERI-extended Ibex microcontroller
  - Small-scale microcontroller used in OpenTitan, etc.
  - CHERI-RISC-V tuned for small microcontrollers
  - Clean-slate memory-safe, compartmentalized embedded OS for high-risk applications
  - Open sourced in February 2023
  - RISC-V embedded standardization candidate
- Collaboration across Microsoft Research, MSRC, Azure Silicon, and Azure Edge + Platform
- lowRISC Sunburst FPGA board reference platform
- Published in IEEE MICRO 2023

lowRISC Sunburst (announced 2023)



- lowRISC-designed/manufactured low-cost FPGA prototyping platform for CHERIoT
- Open consultation on board design and requirements
- Anticipated ship date in 2024
- Supported by UKRI / DSbD

Codasip (announced 2023)



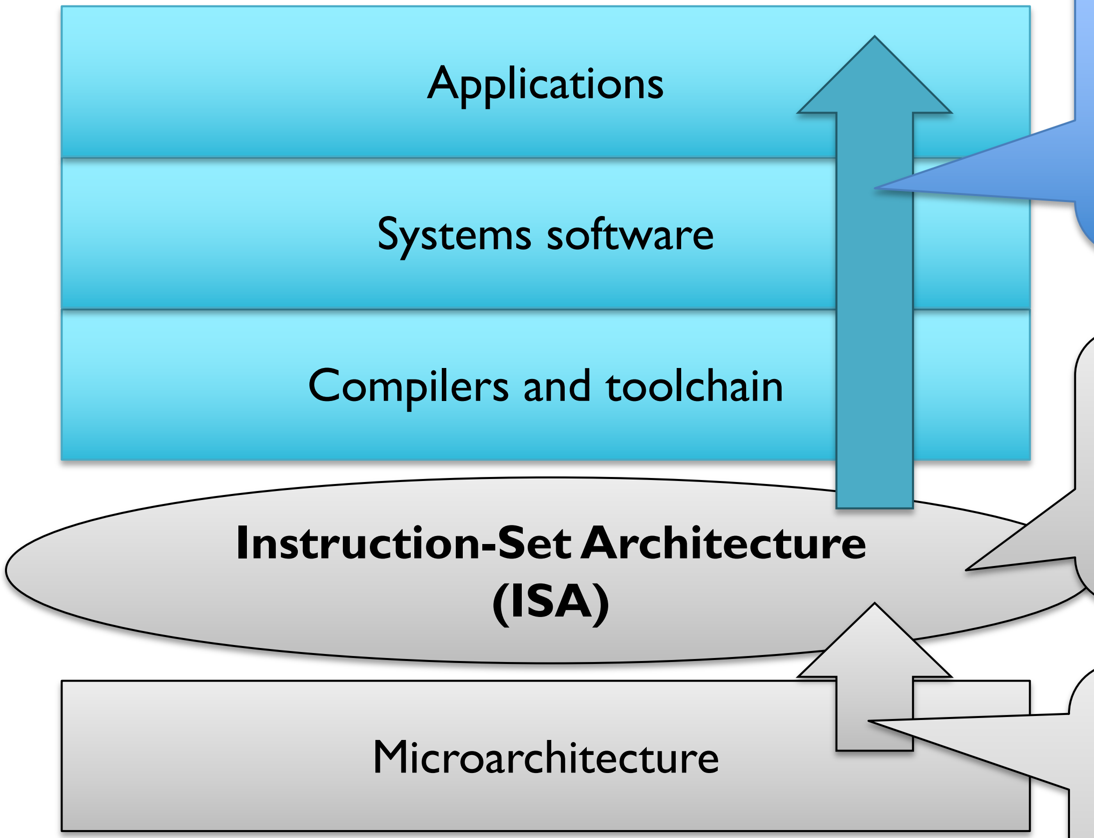
- Commercial CHERI-RISC-V core based on existing RISC-V IP + tooling product
- Codasip is contributing heavily to the CHERI-RISC-V standardization effort
- RISC-V core baseline is pipelined, multicore, MMU-enabled design

RISC-V CHERI SIG and TG

- Ambition: Standardize CHERI use with the RISC-V ISA, given multiple companies building prototypes and products
  - SIG created in October 2022, SIG chair is Alex Richardson (Google), co-chair Simon Moore (Cambridge)
  - TG created in January 2024, same acting chairs
- SIG has been meeting every two weeks for over a year working through use cases, implications for different microarchitecture, various RISC-V standardization considerations
  - First draft specification on verge of being released for community discussion, review, extension
- First CHERI-RISC-V products won't conform as standard not complete, but working hard with industrial partners (e.g., Codasip, Google) to ensure useful convergence

HOW SOFTWARE WORKS ON 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**

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

Early questions:

- Efficient fine-grained architectural memory protection enforces:

Provenance validity:	Q: Where do pointers come from?
Integrity:	Q: How do pointers move in practice?
Bounds, permissions:	Q: What rights should pointers carry?
Monotonicity:	Q: Can real software play by these rules?

More recent questions: CHERI implications for software?

- But also higher-level protection properties:

Heap temporal memory safety Q: Do applications use – or compare pointers after free (e.g., for lockless algorithms)?

Safety for custom allocators Q: Can application-specific allocators also benefit from spatial and temporal safety?

Robustness for code generation Q: Can software that intentionally introduces new code – kernels, run-time linkers, language runtimes – benefit?

Safe isolation and communication Q: Can mutually distrusting software modules communicate safely across strong boundaries?

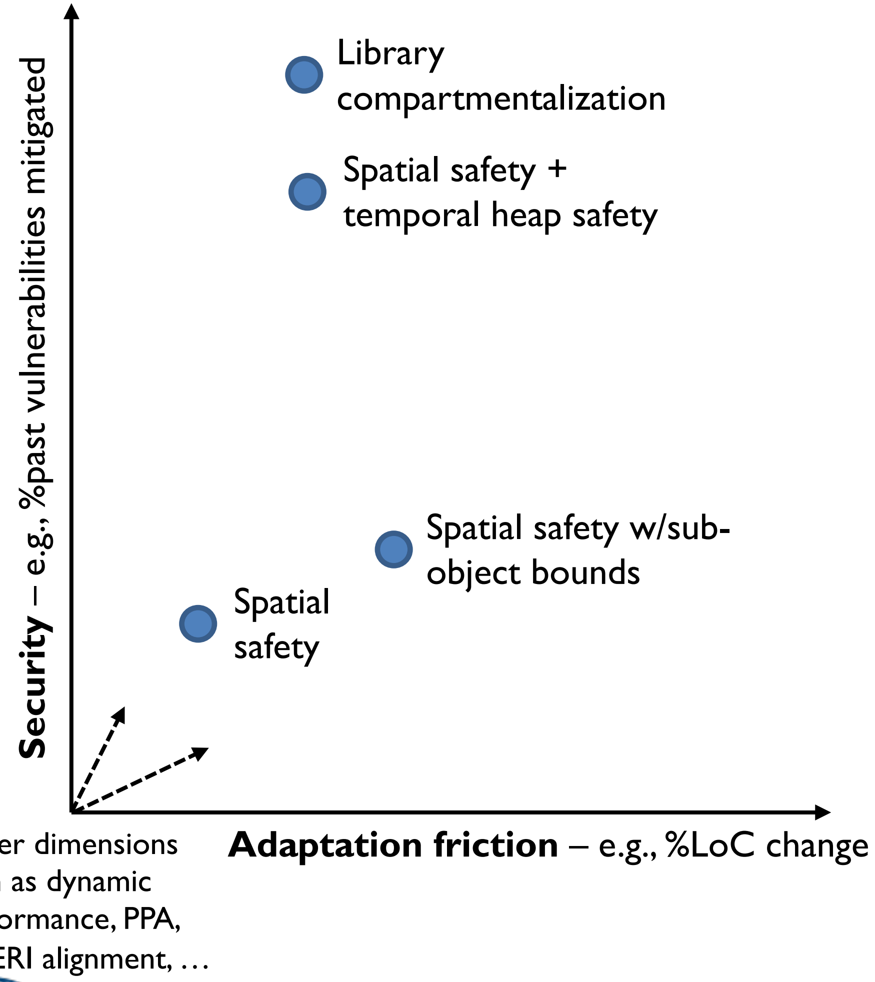
What do we mean by C/C++ memory safety?

- 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/++ 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 exploiting a lack of memory safety
  - **Deterministic mitigation** means that defenses always work regardless of information leakage, attempts to brute force, and so on

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

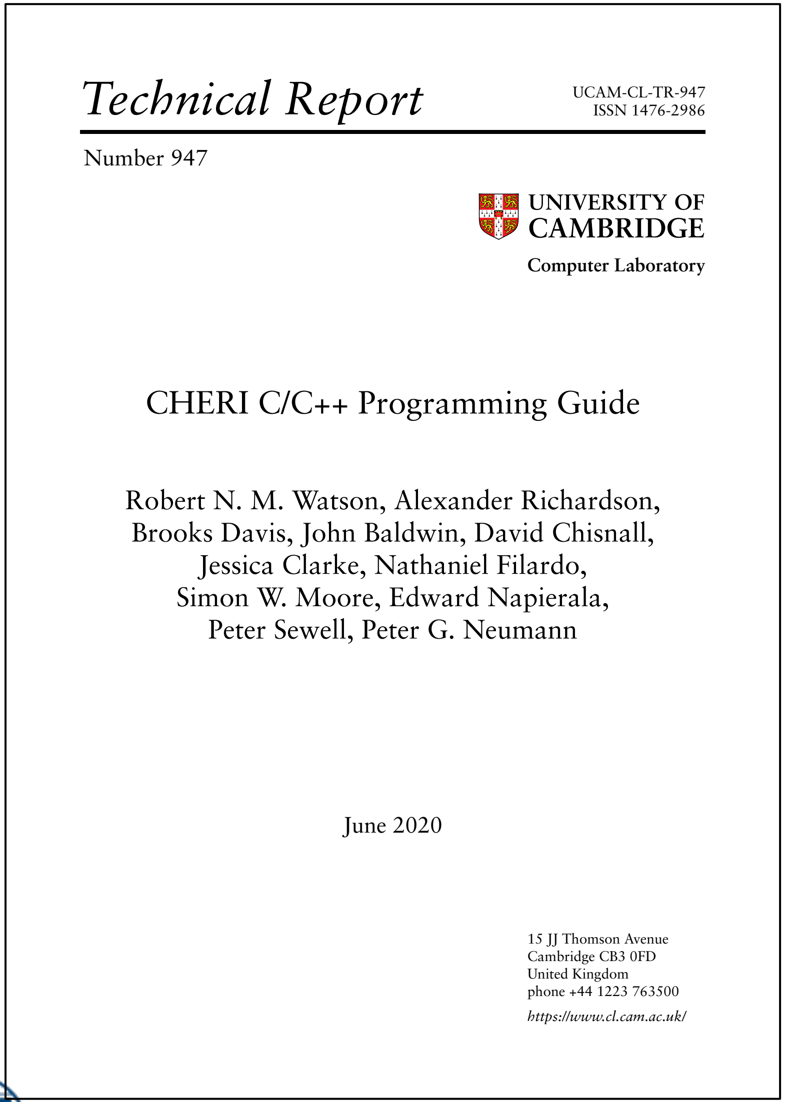
- Our ambition for CHERI C/C++ memory safety is to **mitigate the vast majority (>70%) of memory-safety vulnerabilities with full determinism**

A space of C memory-protection models



- 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

Memory protection for the language and the language runtime

Language-level memory safety

Pointers to heap allocations

Function pointers

Pointers to global variables

Pointers to memory mappings

Pointers to stack allocations

Pointers to TLS variables

Pointers to sub-objects

Return addresses

GOT pointers

Vararg array pointers

PLT entry pointers

Stack pointers

C++ v-table pointers

ELF aux arg pointers

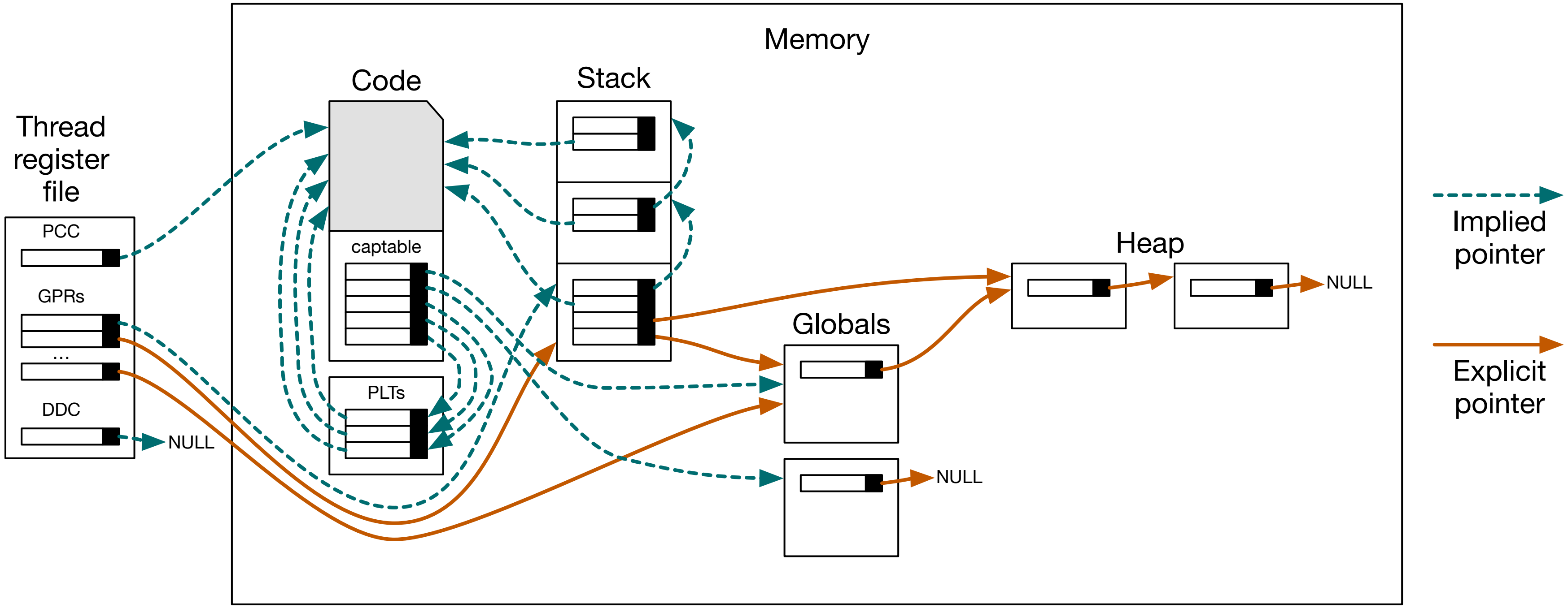
Sub-language memory safety

- Capabilities are refined by the kernel, run-time linker, compiler-generated code, heap allocator, …
- Protection mechanisms:
  - Referential memory safety
  - Spatial memory safety + privilege minimization
  - Temporal memory safety
- Applied **automatically** at two levels:
  - **Language-level pointers** point explicitly at stack and heap allocations, global variables, …
  - **Sub-language pointers** used to implement control flow, linkage, etc.
- Sub-language protection mitigates bugs in the language runtime and generated code, as well as attacks that cannot be mitigated by higher-level memory safety
  - (e.g., union type confusion)

SRI International

UNIVERSITY OF CAMBRIDGE

CHERI-based pure-capability process memory



- Capabilities are substituted for integer addresses throughout the address space
- Bounds and permissions are minimized by software including the kernel, run-time linker, memory allocator, and compiler-generated code
- Hardware permits fetch, load, and store only through granted capabilities
- Tags ensure integrity and provenance validity of all pointers

SRI International

UNIVERSITY OF CAMBRIDGE

RISC-V vs. CHERI-RISC-V generated code

```
struct timezone tz;

time_t get_unix_time(void)
{
    struct timeval tv;

    gettimeofday(&tv, &tz);
    return tv.tv_sec;
}
```

```
get_unix_time_riscv:
 addi    sp, sp, -32
 sd ra, 24(sp)
 addi    a0, sp, 8
.LBB0_1:
 auipc   a1, %pcrel_hi(tz)
 addi    a1, a1, %pcrel_lo(.LBB0_1)
 call    gettimeofday
  (expands to auipc, possibly cld, cjalr)
 ld  a0, 8(sp)
 ld  ra, 24(sp)
 addi    sp, sp, 32
 ret
```

```
get_unix_time_cheririscv:
 cincoffset  csp, csp, -32
 csc     cra, 16(csp)
 cincoffset  ca0, csp, 0
 csetbounds  ca0, ca0, 16
.LBB0_1:
 auipcc  ca1, %captab_pcrel_hi(tz)
 clc ca1, %pcrel_lo(.LBB0_1)(ca1)
.LBB0_2:
 auipcc  ca2, %captab_pcrel_hi(gettimeofday)
 clc ca2, %pcrel_lo(.LBB0_2)(ca2)
 cjalr   cra, ca2
 cld     a0, 0(csp)
 clc     cra, 16(csp)
 cincoffset  csp, csp, 32
 cret
```

1. Adjust stack address/capability
2. Save return address/capability
3. Create address/capability to local 'tv'

4. Generate address/capability to global 'tz'

5. Call gettimeofday()

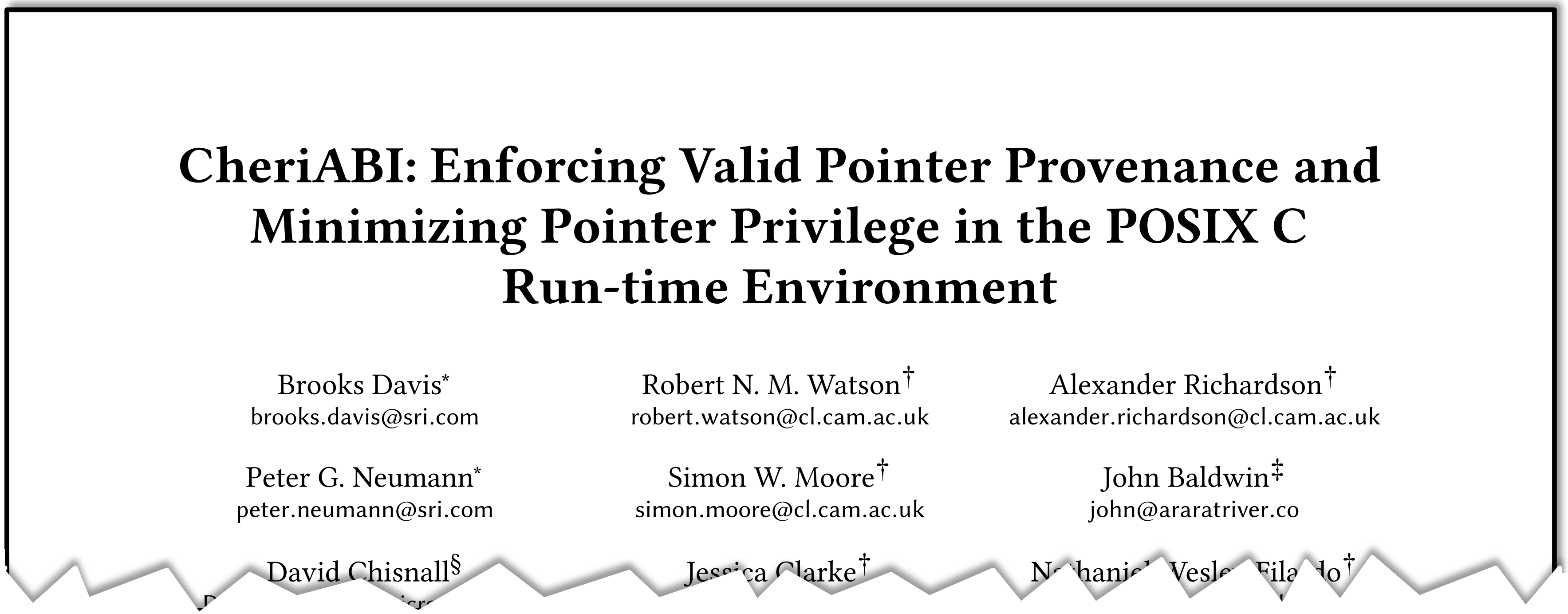
6. Load return value from 'tv'
7. Load return address/capability
8. Restore stack address/capability
9. Return

- The general code structure is unchanged except that:
  - The integer stack pointer becomes a capability stack pointer
  - The pointer to a local stack allocation becomes capability
  - Compiler-specified bounds are set on the local variable pointer before use
  - The loaded jump target is a capability rather than an integer address

CheriBSD: A pure-capability operating system

- Complete memory- and pointer-safe FreeBSD C/C++ kernel + userspace
  - **OS kernel**: Core OS kernel, filesystems, networking, device drivers, …
  - **System libraries**: crt/csu, ld-elf.so, libc, zlib, libxml, libssl, …
  - **System tools and daemons**: echo, sh, ls, openssl, ssh, sshd, …
  - **Applications**: PostgreSQL, nginx, WebKit (C++)
- **Valid provenance**, **minimized privilege** for **pointers, implied VAs**
  - Userspace capabilities originate in **kernel-provided roots**
  - Compiler, allocators, run-time linker, etc., **refine** bounds and perms
- Trading off **privilege minimization**, **monotonicity**, **API conformance**
  - Typically in memory management – realloc(), mmap() + mprotect()

Pure-capability UNIX process environment


CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment

Brooks Davis*
brooks.davis@sri.com

Robert N. M. Watson†
robert.watson@cl.cam.ac.uk

Alexander Richardson†
alexander.richardson@cl.cam.ac.uk

Peter G. Neumann*
peter.neumann@sri.com

Simon W. Moore†
simon.moore@cl.cam.ac.uk

John Baldwin‡
john@araratriver.co

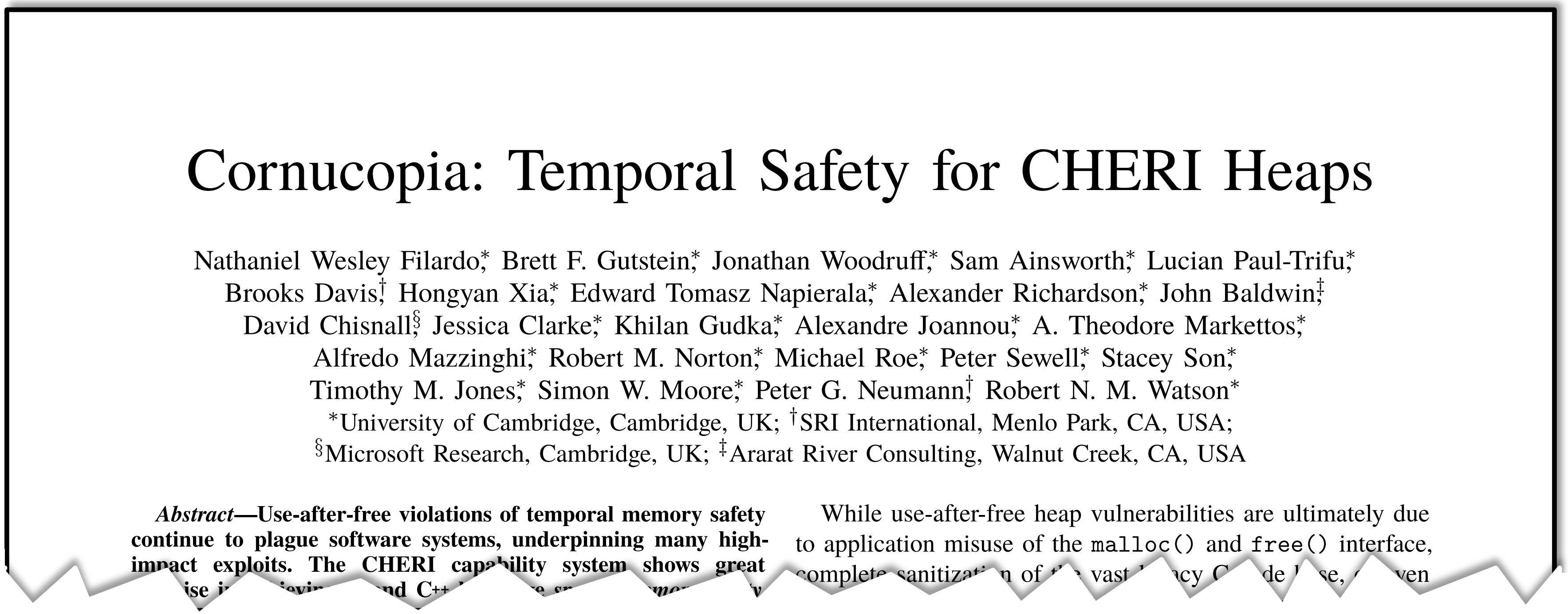
David Chisnall§

Jessica Clarke†

Nathaniel Wesley Filardo†

- Received best paper award at ASPLOS, April 2019
- Complete pure-capability UNIX OS userspace with spatial memory safety
  - Usable for daily development tasks
  - Almost vast majority of FreeBSD tests pass
  - Management interfaces (e.g. ioctl), debugging, etc., work
  - Large, real-world applications have been ported: PostgreSQL and WebKit

Heap temporal memory safety



Cornucopia: Temporal Safety for CHERI Heaps

Nathaniel Wesley Filardo,* Brett F. Gutstein,* Jonathan Woodruff,* Sam Ainsworth,* Lucian Paul-Trifu,*
Brooks Davis,† Hongyan Xia,* Edward Tomasz Napierala,* Alexander Richardson,* John Baldwin,‡
David Chisnall,§ Jessica Clarke,* Khilan Gudka,* Alexandre Joannou,* A. Theodore Markettos,*
Alfredo Mazzinghi,* Robert M. Norton,* Michael Roe,* Peter Sewell,* Stacey Son,*
Timothy M. Jones,* Simon W. Moore,* Peter G. Neumann,† Robert N. M. Watson*
*University of Cambridge, Cambridge, UK; †SRI International, Menlo Park, CA, USA;
§Microsoft Research, Cambridge, UK; ‡Ararat River Consulting, Walnut Creek, CA, USA

***Abstract*—Use-after-free violations of temporal memory safety continue to plague software systems, underpinning many high-impact exploits. The CHERI capability system shows great**

While use-after-free heap vulnerabilities are ultimately due to application misuse of the `malloc()` and `free()` interface,

- IEEE Symposium on Security and Privacy (“Oakland”), May 2020
- Hardware and software support for deterministic temporal memory safety for C/C++-language heaps using capability revocation
- Hardware enables fast tag searching using MMU-assisted tracking of tagged values, tag controller and cache

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

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited

Annona: Load Barriers for CHERI Heap Temporal Safety

Nathaniel Wesley Filardo, Microsoft
Brett F. Gutstein, University of Cambridge
Jonathan Woodruff, University of Cambridge
Jessica Clarke, University of Cambridge
Peter Rugg, University of Cambridge
Brooks Davis, University of Cambridge
Mark Johnston, University of Cambridge
Robert Norton-Wright, Microsoft
David Chisnall, Microsoft
Simon W. Moore, University of Cambridge
Peter G. Neumann, SRI
Robert N. M. Watson, University of Cambridge

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 tantalized heap tempo-ral *safety as well. However, these efforts have sizable CPU and DRAM traffic overheads and significant "stop-the-world" pause times.*

We present Annona, 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 core and CHERI-RISC-V – to nearly eliminate application pauses. We analyze the performance of Annona as well as (re-implementations of) Cornucopia and CHERIvoke on Morello, using the CHERI-compatible SPEC CPU2006 INT workloads to assess its impact on batch workloads and using pgbench *as a representative interactive workload. We find that Annona achieves its goals: applications no longer experience significant revocation-induced world-stopped periods, the system incurs no additional wall- or CPU-time cost relative to Cornucopia, and, pleasantly, this new approach* reduces *the total DRAM traffic used by revocation by a median of 12% across SPEC CPU2006 benchmarks and by over 50% for* pgbench.

1. Introduction

Programming languages, broadly, offer an object-centric (let us not say "oriented") model of memory. New objects, which are 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, for example, …

- confuse integers, object references, and memory indices that do not point to valid objects (such as those used internally by the memory allcoator), risking *reference integrity* violations;
- access adjacent objects, reaching beyond the bounds of a referenced object, violating *spatial safety*;
- 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 significant threat to software security [11, 25], and a wide variety of languages, compilation approaches, and runtime strategies have emerged in response.

The CHERI [40] capability architecture, summarized in §2.1, has shown promise as a technology for C and C++ language reference integrity and spatial safety, with overheads acceptable for general-purpose computing [39]. Strategies for heap temporal safety atop CHERI have emerged, most notably CHERIvoke [44] and its successor Cornucopia [17], and have hinted at viability of a *sweeping revocation* approach (§2.2). However, while Cornucopia's aggregate overheads may be tolerable for high-security workloads, its sizable application pause times ("stop-the-world" phases) still likely limit its use to *non-interactive* high-security workloads.

Targeting this shortcoming, we exploited recent extensions to the CHERI architecture and built Annona,[1] a drop-in replacement for Cornucopia's in-kernel component. The key architectural feature is a per-page capability load barrier (§3.2), supporting a fast global enablement (§4.1). Annona uses this, in tandem with an improved form of Cornucopia's *capability dirty* tracking (§4.2), to replace Cornu-

[1] Annona was a divine personification of grain supply to Rome and was frequently shown holding a cornucopia.

1

- **Cornucopia heap temporal safety** (IEEE SSP 2020), is a GC-inspired, quarantining technique
  - The kernel virtual-memory subsystem 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 10s of microseconds
- Enabled by default in CheriBSD 23.11

UNIVERSITY OF CAMBRIDGE

Ongoing temporal memory-safety deployment

- Shipped in CheriBSD 23.11 release
  - Experimenting with larger-scale software, such as desktop stack
  - Enabled by default in 23.11 to gain exposure; easy to disable
- Looking for increased experience:
  - Semantic impact on any applications vs. bugs/vulnerabilities discovered
  - Acceptability of performance behavior, optimization opportunities
  - Use in higher-level allocators – e.g., APR, Chromium, etc.
  - Support for strong isolation needed for compartmentalization
  - Enabling safe inter-compartment communication via shared memory

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

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

Vadim Zaliva, University of Cambridge, Cambridge, UK, Vadim.Zaliva@cl.cam.ac.uk
Kayvan Memarian, University of Cambridge, Cambridge, UK, Kayvan.Memarian@cl.cam.ac.uk
Ricardo Almeida, University of Edinburgh, Edinburgh, UK, Ricardo.Almeida@ed.ac.uk
Jessica Clarke, University of Cambridge, Cambridge, UK, Jessica.Clarke@cl.cam.ac.uk
Brooks Davis, SRI International, Menlo Park, CA, USA, Brooks.Davis@sri.com
Alexander Richardson, University of Cambridge, Cambridge, UK, Alexander.Richardson@cl.cam.ac.uk
David Chisnall, Microsoft, Cambridge, UK, David.Chisnall@microsoft.com
Brian Campbell, University of Edinburgh, Edinburgh, UK, Brian.Campbell@ed.ac.uk
Ian Stark, University of Edinburgh, Edinburgh, UK, Ian.Stark@ed.ac.uk
Robert N. M. Watson, University of Cambridge, Cambridge, UK, Robert.Watson@cl.cam.ac.uk
Peter Sewell, University of Cambridge, Cambridge, UK, Peter.Sewell@cl.cam.ac.uk

Abstract

Memory safety issues are a persistent source of security vulnerabilities, with conventional architectures and the C codebase chronically prone to exploitable errors. The CHERI research project has shown how one can provide radically improved security for that existing codebase with minimal modification, using unforgeable hardware capabilities in place of machine-word pointers in CHERI dialects of C, implemented as adaptions of Clang/LLVM and GCC. CHERI was first prototyped as extensions of MIPS and RISC-V; it is currently being evaluated by Arm and others with the Arm Morello experimental architecture, processor, and platform, to explore its potential for mass-market adoption, and by Microsoft in their CHERIoT design for embedded cores.

There is thus considerable practical experience with CHERI C implementation and use, but exactly what CHERI C's semantics is (or should be) remains an open question. In this paper, we present the first attempt to rigorously and comprehensively define CHERI C semantics, discuss key semantics design questions relating to capabilities, provenance, and undefined behaviour, and clarify them with semantics in multiple complementary forms: in prose, as an executable semantics adapting the Cerberus C semantics, and mechanised in Coq.

This establishes a solid foundation for CHERI C, for those porting code to it, for compiler implementers, and for future semantics and verification.

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0372-0/24/04.
https://doi.org/10.1145/3617232.3624859

ACM Reference Format:
Vadim Zaliva, Kayvan Memarian, Ricardo Almeida, Jessica Clarke, Brooks Davis, Alexander Richardson, David Chisnall, Brian Campbell, Ian Stark, Robert N. M. Watson, and Peter Sewell. 2024. Formal Mechanised Semantics of CHERI C: Capabilities, Undefined Behaviour, and Provenance. In *28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3617232.3624859

1 Introduction

Memory safety bugs continue to be a major source of security vulnerabilities, despite much research on software bug-finding and mitigation approaches. For example, they are responsible for most of those addressed by Microsoft security updates or impacting Chromium [19, 29]. They are a particular concern for the large codebases in C and C++ that comprise the infrastructure that we all depend on. Alternative memory-safe languages offer promise, but these C/C++ codebases will clearly be an ongoing challenge for the foreseeable future.

- Research paper on a formal semantics and behaviour of CHERI C:
  - CHERI C Semantics Design Questions
  - CHERI C executable semantics
  - Validation and Experimental Comparison
- Considers topics such as, "What are compiler optimizations allowed to do when they recognize undefined behavior such as out-of-bounds accesses"

UNIVERSITY OF CAMBRIDGE

CHERI C compatibility: CheriBSD Code Changes

Area	Files total	Files modified	% files	LoC total	LoC changed	% LoC
Kernel	**11,861**	**896**	**7.6**	**6,095k**	**6,961**	**0.18**
• Core	7,867	705	9.0	3,195k	5,787	**0.18**
• Drivers	3,994	191	4.8	2,900k	1,174	**0.04**
Userspace	**16,968**	**649**	**3.8**	**5,393k**	**2,149**	**0.04**
• Runtimes (excl. libc++)	1,493	233	15.6	207k	989	**0.48**
• libc++	227	17	7.5	114k	133	**0.12**
• Programs and libraries	15,475	416	2.7	5,186k	1,160	**0.02**

Notes:

- Numbers from cloc counting modified files and lines for identifiable C, C++, and assembly files
- Kernel includes changes to be a hybrid program and most changes to be a pure-capability program
  - Also includes most of support for CHERI-MIPS, CHERI-RISC-V, Morello
  - Count includes partial support for 32 and 64-bit FreeBSD and Linux binaries.
  - 67 files and 25k LoC added to core in addition to modifications
  - Most generated code excluded, some existing code could likely be generated

Pure-capability CheriBSD kernel

- Full UNIX operating-system kernel compiled with CHERI C
  - Roughly 2.4MLoC core kernel excluding device drivers
  - Referential safety for all explicit and implied pointers
  - Spatial safety for mappings, stack and heap allocations, globals; with sub-object bounds
  - Temporal memory safety is not yet supported, work is being planned.
- 1.4% LoC change, 7.7% files changed
  - Includes support for hybrid kernel with CheriABI userspace, which requires capability annotations for system-call arguments
  - **We will have better data on a pure purecap kernel soon, stripping hybrid support, which should substantially reduce %LoC change**

Pure-capability CheriBSD kernel: Vulnerabities

- Security analysis based on retrospective vulnerability study over 22 years
- 56% of total vulnerabilities (113 of 200) are memory-safety; of these:
    - 54% mitigated through referential and spatial safety (implemented); of these, 8% of memory safety w/sub-object
    - 72% mitigated if including heap temporal memory safety (white-board design)
    - 26% unmitigated are uninitialized values; at least 5% of memory safety would likely be mitigated by LLVM stack initialization
    - Handful of unmitigated vulnerabilities: stack temporal safety, VM vulnerabilities, …

  %s are of memory-safety vulnerabilities
- 1 FTE for ~2.5 years for MIPS, RISC-V, and Morello; most time on common code
- Be aware of selection bias in vulnerability discovery – e.g., KASAN finding use-after-free vulnerabilities with fuzzing, but not subobject bounds overflows

SRI International

UNIVERSITY OF CAMBRIDGE

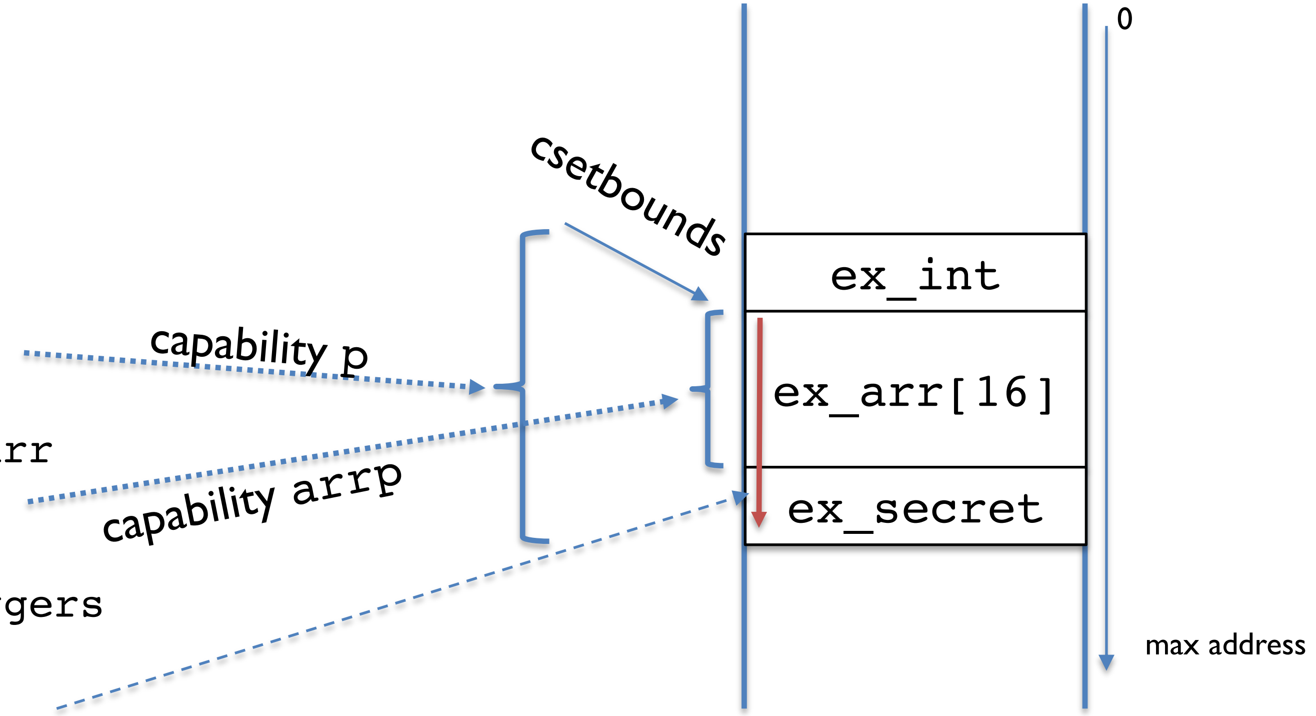
Pure-capability CheriBSD kernel: Sub-object bounds

```
struct example {
    int ex_int;
    char ex_arr[16];
    int ex_secret;
};

// Example allocation
struct example *p;
p = malloc(sizeof(*p));

// Narrow bounds on ex_arr
char *arrp = p->ex_arr;

// Overflowing copy triggers
// bounds violation
memcpy(arrp, src, 20);
```



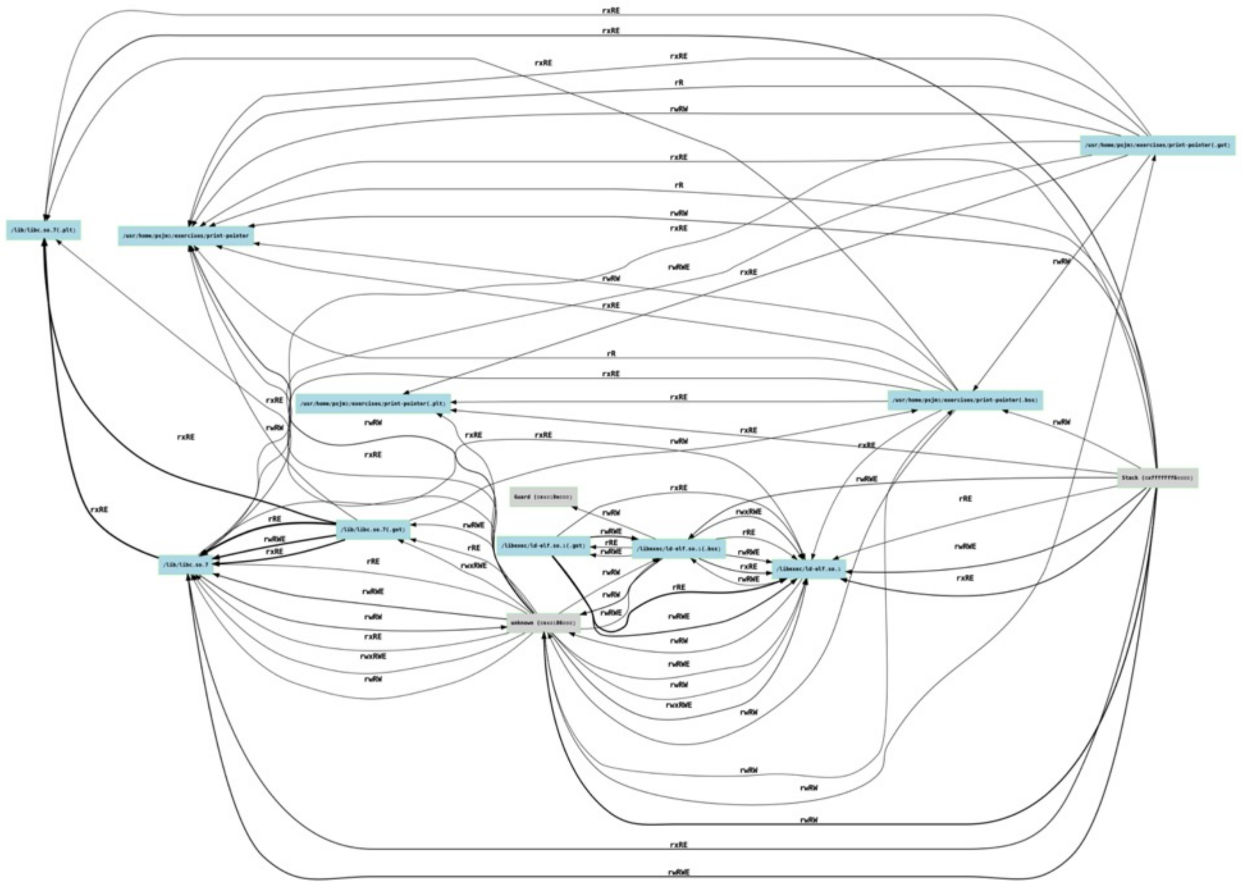
- Sub-object bounds are an optional compilation mode for CHERI C
- Additional protection at slightly greater friction due to `containerof()`

UNIVERSITY OF CAMBRIDGE

Pure-capability CheriBSD kernel: Sub-object bounds

- Automatic bounds narrowing does not cause porting problems in most of the cases:
  - Core kernel requires changes to ~80 files (13% of the core kernel files)
  - Changes consist of simple annotations, magnitude of changes is small.
  - Detecting sub-object bounds incompatibilities can only be done at run-time. Limited by test coverage.
  - Kernel drivers are known to have poor test coverage. Likely that additional changes will be required here.
- Kernel uses "*subobject-safe*" policy for bounds narrowing:
  - Enforces sub-object bounds everywhere except for array indexing
  - In practice this does not affect the ability to mitigate past vulnerabilities

Capability graph visualization and analysis



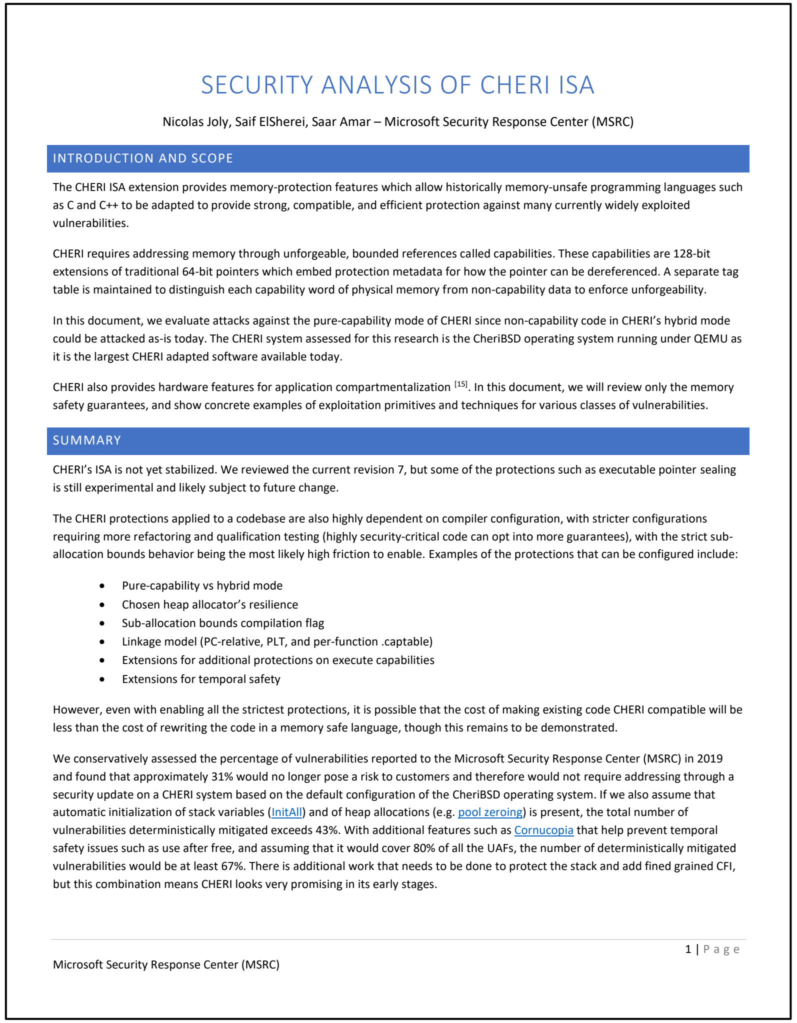
- Pointers are now directly visible in hardware – in memory, ISA-level traces, and so on
  - We can directly analyze capability delegation with CHERI
- New extraction tools scan virtual addresses spaces and binaries to enable:
  - Visualization
  - Validation
  - Debugging and optimization
- **Allows direct analysis of attacker-visible resources and attack surfaces**

pp.db

START	END	PRT	ro	rw	rx	rwx	TOTAL	DENSITY	FLAGS	TP	PATH
0x101000	0x110000	-----	0	0	0	0	0	0.00	-----	--	print-pointer
0x110000	0x111000	r-xR-	0	0	0	0	0	0.00	CN---	vn	print-pointer(.plt)
0x111000	0x120000	-----	0	0	0	0	0	0.00	-----	--	print-pointer
0x120000	0x122000	r--R-	9	6	6	0	21	0.24	C----	vn	print-pointer(.got)
0x122000	0x131000	-----	0	0	0	0	0	0.00	-----	--	print-pointer
0x131000	0x132000	rw-RW	1	5	8	0	14	0.16	-----	sw	print-pointer(.bss)
0x40131000	0x40139000	r--R-	0	0	0	0	0	0.00	CN---	vn	ld-elf.so.1
0x40139000	0x40148000	-----	0	0	0	0	0	0.00	-----	--	ld-elf.so.1
0x40148000	0x40162000	r-xR-	0	0	0	0	0	0.00	CN---	vn	ld-elf.so.1
0x40162000	0x40171000	-----	0	0	0	0	0	0.00	-----	--	ld-elf.so.1
0x40171000	0x40174000	rw-RW	462	89	5	0	556	6.37	C----	vn	ld-elf.so.1(.got)
0x40174000	0x40183000	-----	0	0	0	0	0	0.00	-----	--	ld-elf.so.1
0x40183000	0x40184000	rw-RW	32	7	0	0	39	0.45	C----	vn	ld-elf.so.1
0x40184000	0x40186000	rw-RW	3	36	21	11	71	0.81	-----	sw	ld-elf.so.1(.bss)
0x40186000	0x4018e000	rw-RW	0	316	4	10	330	3.78	-----	sw	unknown
0x4018e000	0x4018f000	-----	0	0	0	0	0	0.00	CN---	gd	Guard
0x40190000	0x40221000	r--R-	0	0	0	0	0	0.00	CN---	vn	libc.so.7
0x40221000	0x40230000	-----	0	0	0	0	0	0.00	CN---	gd	libc.so.7
0x40230000	0x4035e000	r-xR-	0	0	0	0	0	0.00	CN---	vn	libc.so.7(.plt)
0x4035e000	0x4036e000	-----	0	0	0	0	0	0.00	CN---	gd	libc.so.7
0x4036e000	0x4038a000	r--R-	3643	699	537	0	4879	55.92	C----	vn	libc.so.7(.got)
0x4038a000	0x40399000	-----	0	0	0	0	0	0.00	CN---	gd	libc.so.7
0x40399000	0x403a4000	rw-RW	127	49	1151	0	1327	15.21	C----	vn	libc.so.7
0x403a4000	0x407dd000	rw-RW	1	269	4	0	274	3.14	-----	sw	libc.so.7
0x407dd000	0x407fe000	rw-RW	0	6	0	0	6	0.07	-----	sw	unknown
0x407fe000	0x40805000	-----	0	0	0	0	0	0.00	CN---	gd	Guard
0x40805000	0x40815000	rw-RW	0	57	0	0	57	0.65	-----	sw	unknown
0x40815000	0x40a15000	rw-RW	0	210	0	0	210	2.41	-----	sw	unknown
0x60000000	0x60200000	rw-RW	1	140	0	0	141	1.62	-----	sw	unknown
0x80000000	0x80600000	rw-RW	0	22	0	0	22	0.25	-----	sw	unknown
0xffffbff00000	0xffffbff80000	rw-RW	1	28	1	1	31	0.36	-----	sw	unknown
0xffffbff80000	0xffffffff60000	-----	0	0	0	0	0	0.00	-----	gd	Guard
0xffffffff60000	0xffffffff80000	rw-RW	7	640	101	0	748	8.57	---D-	sw	Stack
0xfffffffff000	0x1000000000000	r-x--	0	0	0	0	0	0.00	-----	ph	unknown

```
------------------------------------
pp.db - total number of caps: 6480
------------------------------------
     CAP_LOC  CAP_LOC_SYM (TYPE)                          CAP_INFO                                      CAP_SYM (TYPE)
  0x403709b0  sys_errlist (OBJECT)                        0x401e52c8[rRE,-0x401e52c8]                   SYM NOT FOUND
  0x403709b0  __hidden_sys_errlist (OBJECT)               0x401e52c8[rRE,-0x401e52c8]                   SYM NOT FOUND
  0x403709b0  sys_errlist (OBJECT)                        0x401e52c8[rRE,-0x401e52c8]                   SYM NOT FOUND
  0x403731e0  sys_signame (OBJECT)                        0x401e7ef8[rRE,-0x401e7ef8]                   SYM NOT FOUND
  0x403731e0  sys_signame (OBJECT)                        0x401e7ef8[rRE,-0x401e7ef8]                   SYM NOT FOUND
  0x40373410  sys_siglist (OBJECT)                        0x401e7ef8[rRE,-0x401e7ef8]                   SYM NOT FOUND
  0x40373410  sys_siglist (OBJECT)                        0x401e7ef8[rRE,-0x401e7ef8]                   SYM NOT FOUND
  0x40376e10  __nsdefaultsrc (OBJECT)                     0x401e6804[rRE,-0x401e6804]                   SYM NOT FOUND
  0x40376e10  __nsdefaultsrc (OBJECT)                     0x401e6804[rRE,-0x401e6804]                   SYM NOT FOUND
  0x4037ffa0  __je_arena_dalloc_junk_small (OBJECT)       0x4030ff49[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40386280  __je_extent_hooks_default (OBJECT)          0x4032f59d[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40386390  __je_large_dalloc_junk (OBJECT)             0x40338ca1[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403863a0  __je_large_dalloc_maybe_junk (OBJECT)       0x40338cad[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40386410  __je_nstime_monotonic (OBJECT)              0x4033c37d[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40386420  __je_nstime_update (OBJECT)                 0x4033c385[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40386480  __je_prof_dump_open (OBJECT)                0x4033c869[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40386490  __je_prof_dump_header (OBJECT)              0x4033c8e5[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403865b0  __je_rtree_node_alloc (OBJECT)              0x4033f4d9[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403865c0  __je_rtree_node_dalloc (OBJECT)             0x4033f50d[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403865d0  __je_rtree_leaf_alloc (OBJECT)              0x4033f519[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403865e0  __je_rtree_leaf_dalloc (OBJECT)             0x4033f54d[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40387cb0  __je_witness_lock_error (OBJECT)            0x403561b9[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40387cc0  __je_witness_owner_error (OBJECT)           0x40356251[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40387cd0  __je_witness_not_owner_error (OBJECT)       0x40356285[rxRE,-0x40190000]                  SYM NOT FOUND
  0x40387ce0  __je_witness_depth_error (OBJECT)           0x403562b9[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403992d0  __dso_handle (OBJECT)                       0x403992d0[rwRWE,-0x403992d0]                 __dso_handle (OBJECT)
  0x403992e0  __default_hash (OBJECT)                     0x40237a0d[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403992f0  __thr_jtable (OBJECT)                       0x4023cde1[rxRE,-0x40190000]                  SYM NOT FOUND
  0x403992f0  __thr_jtable (OBJECT)                       0x4023cde1[rxRE,-0x40190000]                  SYM NOT FOUND
  0x4039a0e0  _citrus_NONE_stdenc_ops (OBJECT)            0x40260d29[rxRE,-0x40190000]                  SYM NOT FOUND
  0x4039a180  _citrus_stdenc_default (OBJECT)             0x4039a0e0[rwRWE,-0x4039a0e0]                 _citrus_NONE_stdenc_ops (OBJECT)
  0x4039b510  _CurrentRuneLocale (OBJECT)                 0x401eadb0[rwRWE,-0x401eadb0]                 _DefaultRuneLocale (OBJECT)
  0x4039b510  _CurrentRuneLocale (OBJECT)                 0x401eadb0[rwRWE,-0x401eadb0]                 _DefaultRuneLocale (OBJECT)
  0x4039bdc0  h_errlist (OBJECT)                          0x401e592e[rRE,-0x401e592e]                   SYM NOT FOUND
  0x4039bdc0  h_errlist (OBJECT)                          0x401e592e[rRE,-0x401e592e]                   SYM NOT FOUND
  0x4039be10  _res_opcodes (OBJECT)                       0x401e489c[rRE,-0x401e489c]                   SYM NOT FOUND
  0x4039c490  __stdinp (OBJECT)                           0x4039bf20[rwRWE,-0x4039bf20]                 SYM NOT FOUND
  0x4039c490  __stdinp (OBJECT)                           0x4039bf20[rwRWE,-0x4039bf20]                 SYM NOT FOUND
  0x4039c4a0  __stdoutp (OBJECT)                          0x4039c0f0[rwRWE,-0x4039bf20]                 SYM NOT FOUND
  0x4039c4a0  __stdoutp (OBJECT)                          0x4039c0f0[rwRWE,-0x4039bf20]                 SYM NOT FOUND
  0x4039c4b0  __stderrp (OBJECT)                          0x4039c2c0[rwRWE,-0x4039bf20]                 SYM NOT FOUND
  0x4039c4b0  __stderrp (OBJECT)                          0x4039c2c0[rwRWE,-0x4039bf20]                 SYM NOT FOUND
  0x4039c4f0  __sglue (OBJECT)                            0x4039c4c0[rwRWE,-0x4039c4c0]                 SYM NOT FOUND
  0x4039f570  tzname (OBJECT)                             0x4039f560[rwRWE,-0x4039f560]                 SYM NOT FOUND
  0x4039f570  tzname (OBJECT)                             0x4039f560[rwRWE,-0x4039f560]                 SYM NOT FOUND
  0x4039f5a0  __libc_interposing (OBJECT)                 0x402ca3c5[rxRE,-0x40190000]                  __sys_accept (FUNC)
  0x4039f890  svc_auth_null_ops (OBJECT)                  0x402d8541[rxRE,-0x40190000]                  SYM NOT FOUND
  0x4039fa40  __je_opt_junk (OBJECT)                      0x401e148f[rRE,-0x401e148f]                   SYM NOT FOUND
```

MSRC: 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.

- Study analyzed all 2019 critical security vulnerabilities
  - Metric: "Poses a risk to customers → requires a software update"
- 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 build CHERI experience
  - Potential adversarial techniques post-CHERI
  - Recently shifted from CHERI-MIPS to CHERI-RISC-V and Arm Morello

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 Research 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/

SRI International

UNIVERSITY OF CAMBRIDGE

Security Analysis of CHERI ISA

Security Research & Defense / By MSRC Team / October 14, 2020 /
Memory Corruption, Memory Safety, Secure Development, Security Research

Is it possible to get to a state where memory safety issues would be deterministically mitigated? Our quest to mitigate memory corruption vulnerabilities led us to examine CHERI (Capability Hardware Enhanced RISC Instructions), which provides memory protection features against many exploited vulnerabilities, or in other words, an architectural solution that breaks exploits. We've looked at how CHERI would break class-specific categories of vulnerabilities and considered additional mitigations to put in place to get to a comprehensive solution. **We've assessed the theoretical impact of CHERI on all the memory safety vulnerabilities we received in 2019, and concluded that in its current state, and combined with other mitigations, it would have deterministically mitigated at least two thirds of all those issues**.

We've reviewed revision 7 and used CheriBSD running under QEMU as a test environment. In this research, we've also looked for weaknesses in the model and ended up developing exploits for various security issues using CheriBSD and qtwebkit. We've highlighted several areas that warrant improvements, such as vulnerability classes that CHERI doesn't mitigate at the architectural level, the importance of using reliable and CHERI compliant memory management mechanisms, and multiple exploitation primitives that would still allow memory corruption issues to be exploited. **While CHERI does a fantastic job at breaking spatial safety issues, more is needed to tackle temporal and type safety issues**.

Your feedback is extremely important to us as there's certainly much more to discover and mitigate. We're looking forward to your comments on our paper.

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

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

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 the past 6 months, the CHERI project has adapted more lines of open-source code to memory safety than the Rust project has created in its entire history.

* Synopsys Black Duck Open Hub: https://www.openhub.net/languages - Stats taken13 December 2023

SRI International

UNIVERSITY OF CAMBRIDGE

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.

SRI International

UNIVERSITY OF CAMBRIDGE

How should people ask for memory safety?

- Transition appears to be even harder than developing the technology in the first place
- One key challenge is how people can ask for memory safety
  - Poorly satisfied by today's mitigation techniques – stack canaries, PAC, ASLR, …
  - Well satisfied by "up and coming" technologies such as CHERI, Rust, etc.
- How can you request (and be satisfied that you will receive) memory safety on your government procurement form?
- Will require engagement with technical and procedural aspects of the problem
- Host a series of workshops focused on both ..? Your thoughts very welcome!

How does CHERI relate to other **non**-C/C++ memory-safety technologies?

CHERI C/C++

- Requires new, multi-vendor hardware rollout
- Modest changes even to large software packages (Often around ~0.02 %LoC)

  Requires more significant changes to specific packages – e.g., kernels, language runtimes
- The rollout can be done incrementally .. Once there is new hardware
- Dynamic enforcement prevents run-time exploitation – but means that crashes may occur
- Compartmentalization avoids trust in the compiler, handles code generated by adversaries

Rust

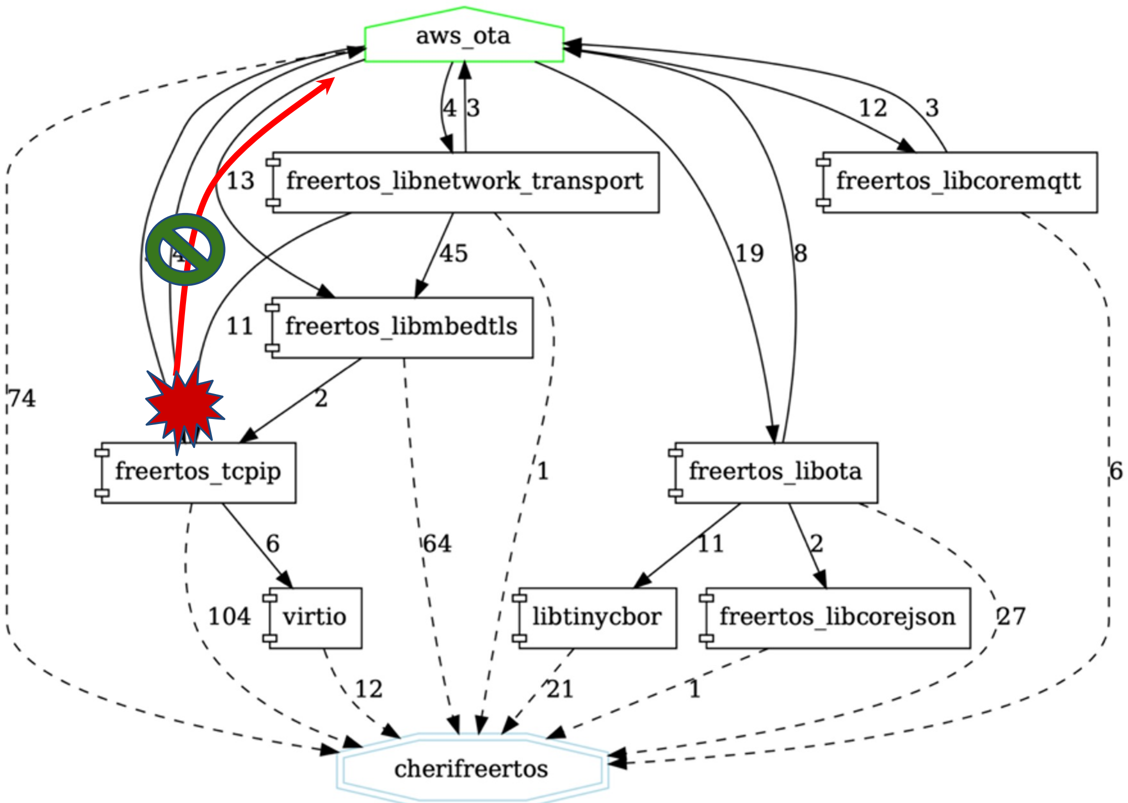
- Requires rewrite of all source code in a new programming language
- Extensive use of "unsafe Rust" can undermines safety for TCBs and in some use cases
- The rollout can be done incrementally .. On current hardware
- Most memory-safety bugs eliminated at compile time, supporting design changes to prevent bugs
- Strong trust in compiler, and no model for handling code generated by an adversary

But what if we put Rust and CHERI together?

- Research question: Is the whole greater than the sum of the parts?
- Lots of reasons to imagine that this might be true, including:
  - Enable a fully memory-safe software ecosystem without 100% software rewrite
  - Reduce total trust in the Rust compiler, enabling downloadable precompiled Apps, device driver sandboxing, …
  - Enforce basic spatial and temporal memory protection for unsafe Rust
  - Use CHERI sub-language protection with Rust to reduce exposure to compiler bugs, new exploit techniques
  - Contain vulnerabilities in C/C++ libraries and other system TCBs
- But ..All of these ideas unimplemented and unevaluated

CHERI SOFTWARE COMPARTMENTALISATION

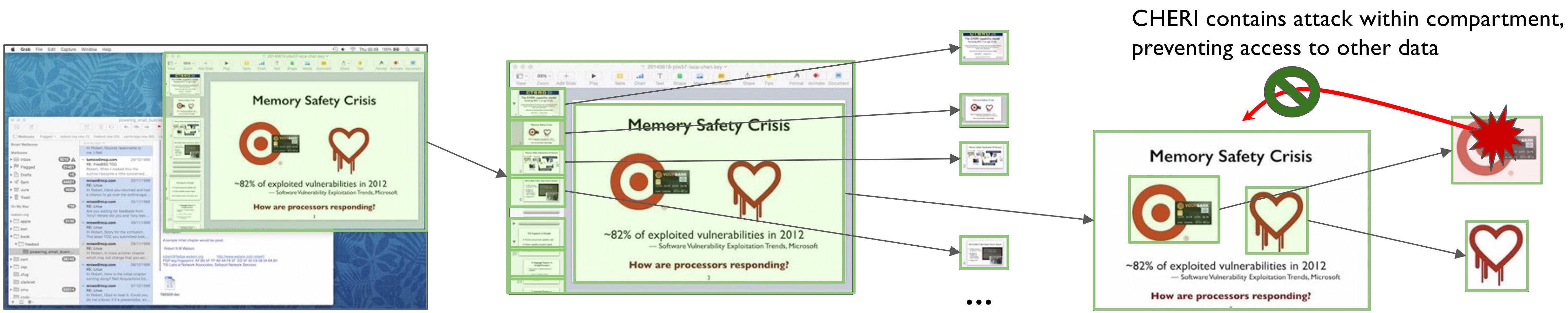
What is software compartmentalization?



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.

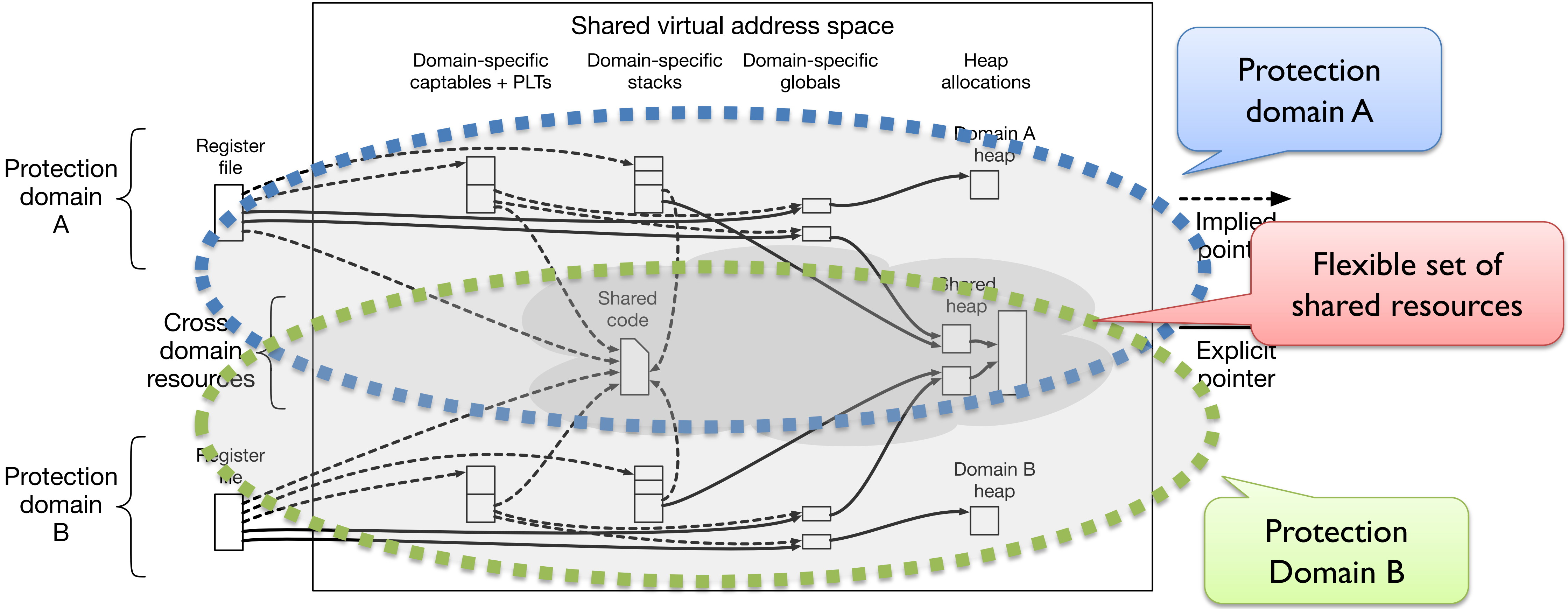
- Fine-grained decomposition of a larger software system into **isolated modules** to constrain the impact of faults or attacks
- Goals is to **minimize privileges yielded by a successful attack, and to limit further attack surfaces**
- Usefully thought about as a **graph of interconnected components**,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

Software compartmentalization at scale



- Current CPUs limit:
  - The number of compartments and rate of their creation/destruction
  - The frequency of switching between them, especially as compartment count grows
  - The nature and performance of memory sharing between compartments
- CHERI is intended to improve each of these – by at least an order of magnitude

CHERI-based compartmentalization



- Isolated compartments can be created using closed graphs of capabilities, combined with a constrained non-monotonic domain-transition mechanism

SRI International

UNIVERSITY OF CAMBRIDGE

Compartmentalization scalability

- CHERI dramatically improves **compartmentalization scalability**
  - More compartments
  - More frequent and faster domain transitions
  - Faster shared memory between compartments

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

- Many potential use cases – e.g., sandbox processing of each image in a web browser, processing each message in a mail application
- Unlike memory protection, software compartmentalization requires **careful software refactoring** to support strong encapsulation, and affects the software operational model

Operational models for CHERI compartmentalization

- An **architectural protection model** enabling new software behavior
- As with virtual memory, multiple **operational models** can be supported
  - E.g., with an MMU: Microkernels, processes, virtual machines, etc.
  - How are compartments created/destroyed? Function calls vs. message passing? Signaling, debugging, …?
- We have explored multiple viable CHERI-based models to date, including:

Isolated dynamic libraries	Efficient but simple sandboxing in processes
UNIX co-processes	Multiple processes share an address space

- Improved performance and new paradigms using CHERI primitives
- Both will be available in CheriBSD/Morello

Proposed operational models: Isolated libraries and UNIX co-processes

Isolated dynamically linked libraries

- New API loads libraries into in-process sandboxes.
- Calling functions in isolated libraries performs a domain transition, with overheads comparable to function calls.
- Simple model eschews asynchrony, independent debugging, etc.

Prototype to appear in CheriBSD 22.10; updates in 23.10

UNIX co-processes

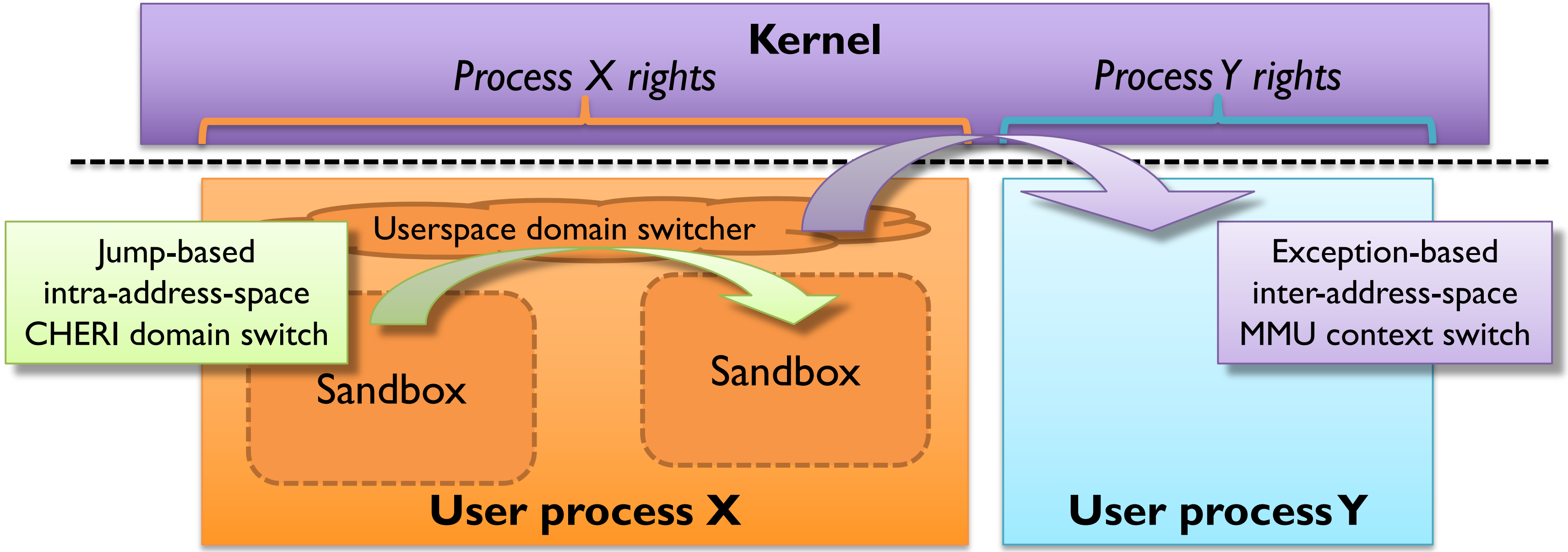
- Multiple processes share a single virtual address space, separated using independent CHERI capability graphs.
- CHERI capabilities enable efficient sharing, domain transition.
- Rich model associates UNIX process with each compartment.

Prototype to appear in future CheriBSD release

SRI International

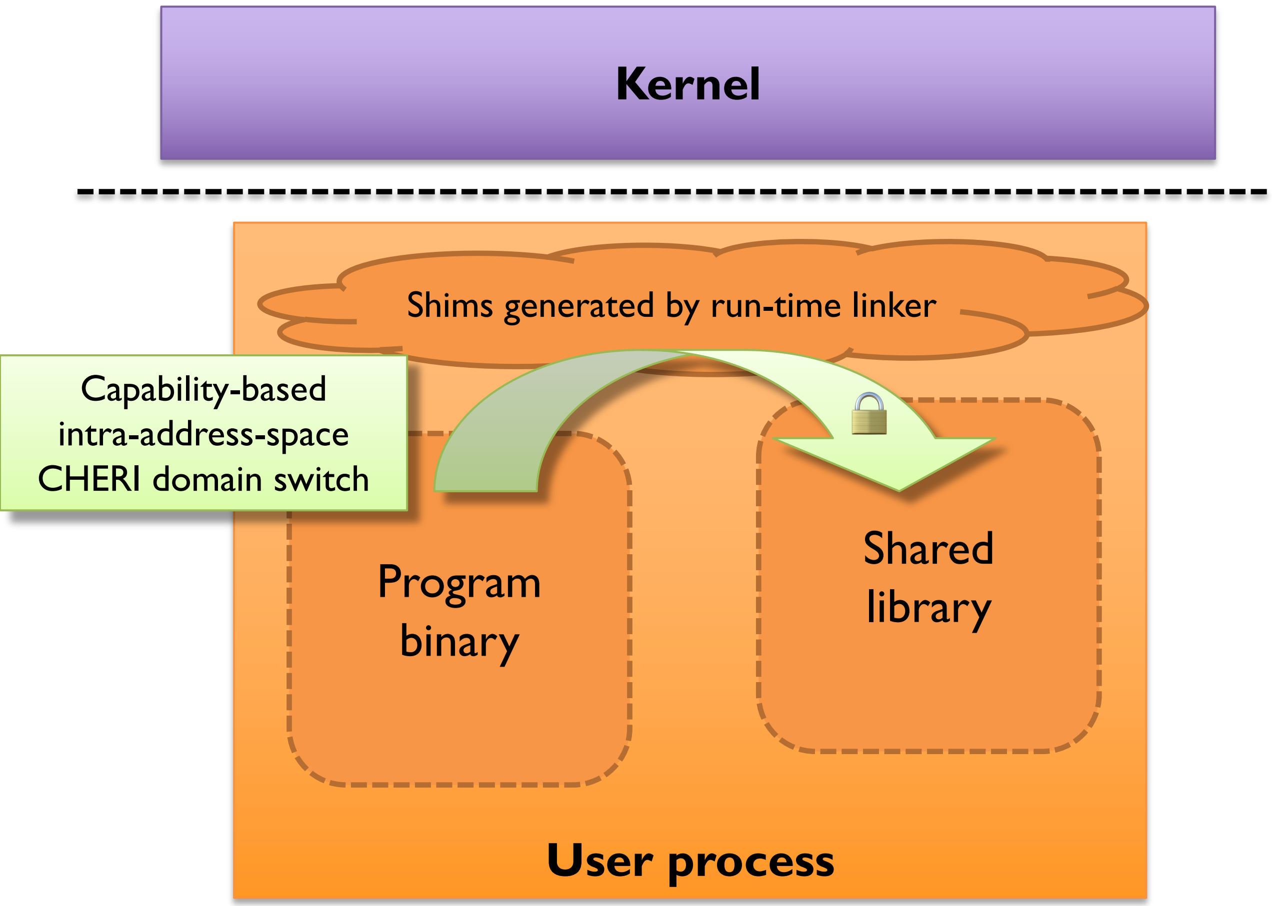
UNIVERSITY OF CAMBRIDGE

Example: Robust shared libraries



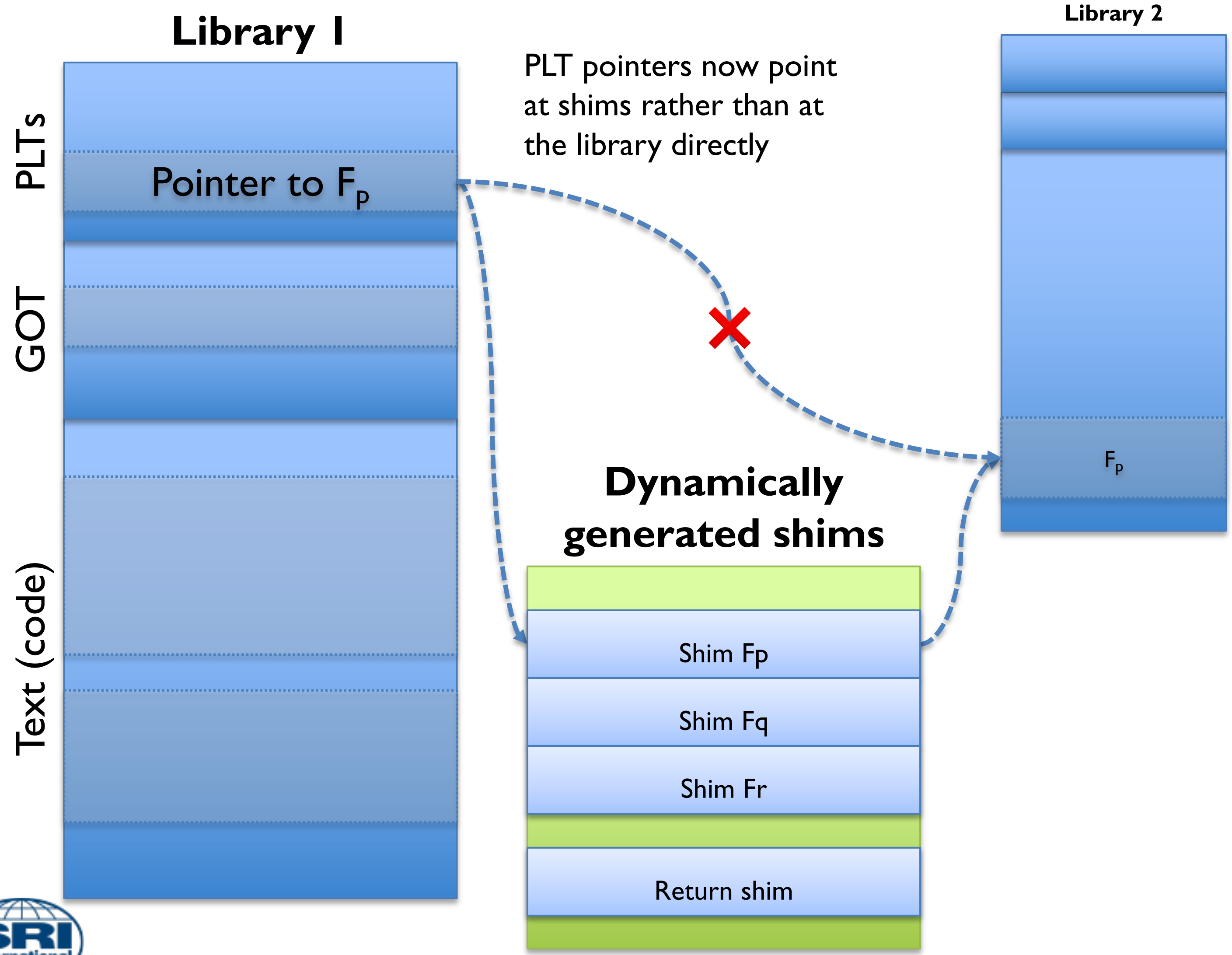
- User compartments exist **within individual UNIX processes** ("robust shared libraries"):
  - CHERI isolates compartments within each address spaces
  - Compartment switcher is itself a trusted userspace library
  - Compartments have strict subset of OS rights of the process
- Intra-process domain switches take **no architectural exceptions** and **do not enter the kernel**
- Multiple processes + IPC required if differing OS right sets needed

Shared library compartmentalization (1/3)



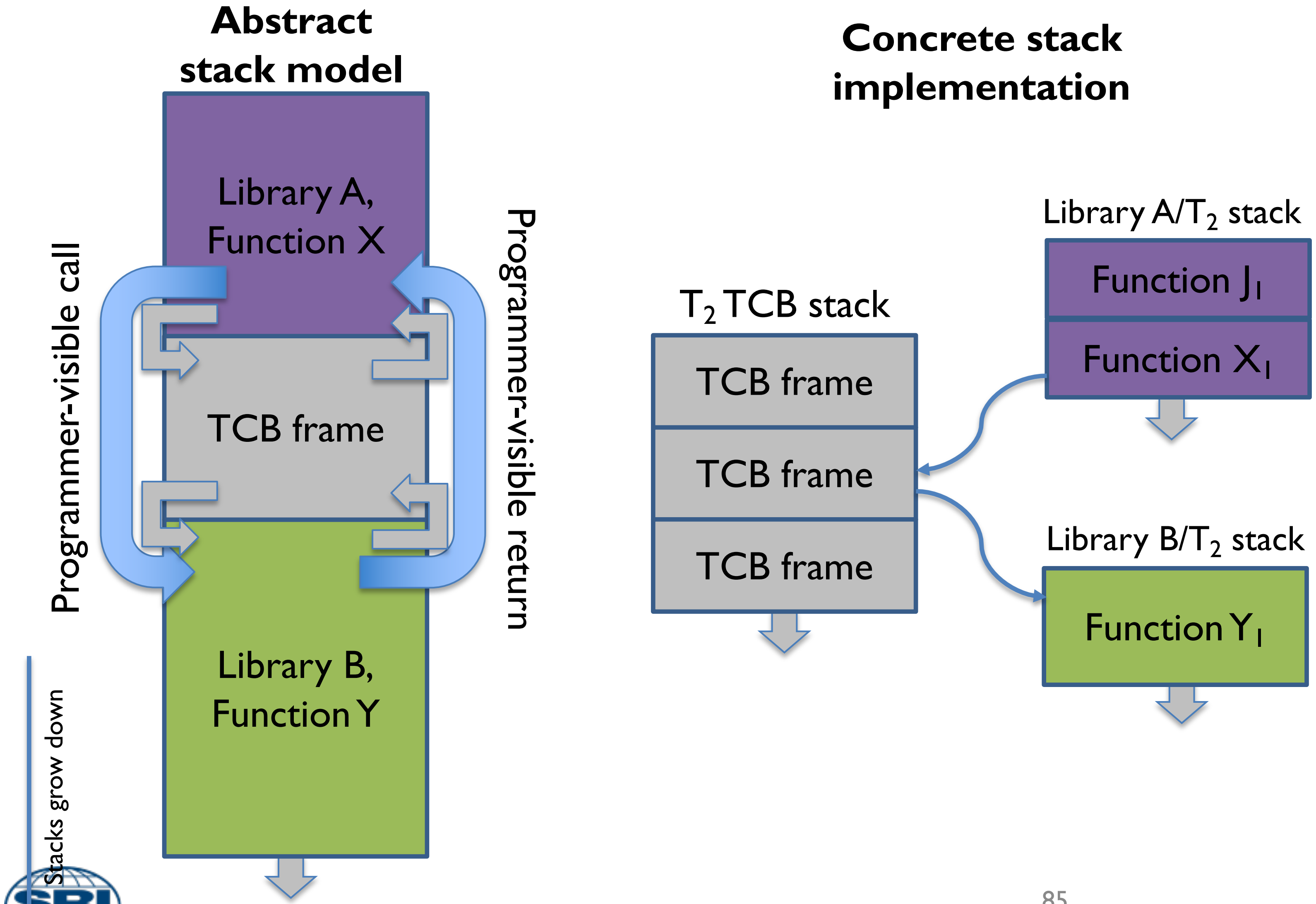
- Run-time linker limits shared libraries to accesses enabled by ELF
  - Adversary model assumes arbitrary code execution within library
  - Run-time linker delegates capabilities for linked functions, globals via GOT/PLT
  - Domain transitions implemented by trampolines interposed on inter-object calls / returns
- Running prototype on Arm Morello
  - Low measured overheads in early experiments (e.g., ~1% for image decompression sandboxing)
  - Released in CheriBSD 22.12 in December
  - Debugging, tracing, and performance enhancements in CheriBSD 23.10

Shared library compartmentalization (2/3)



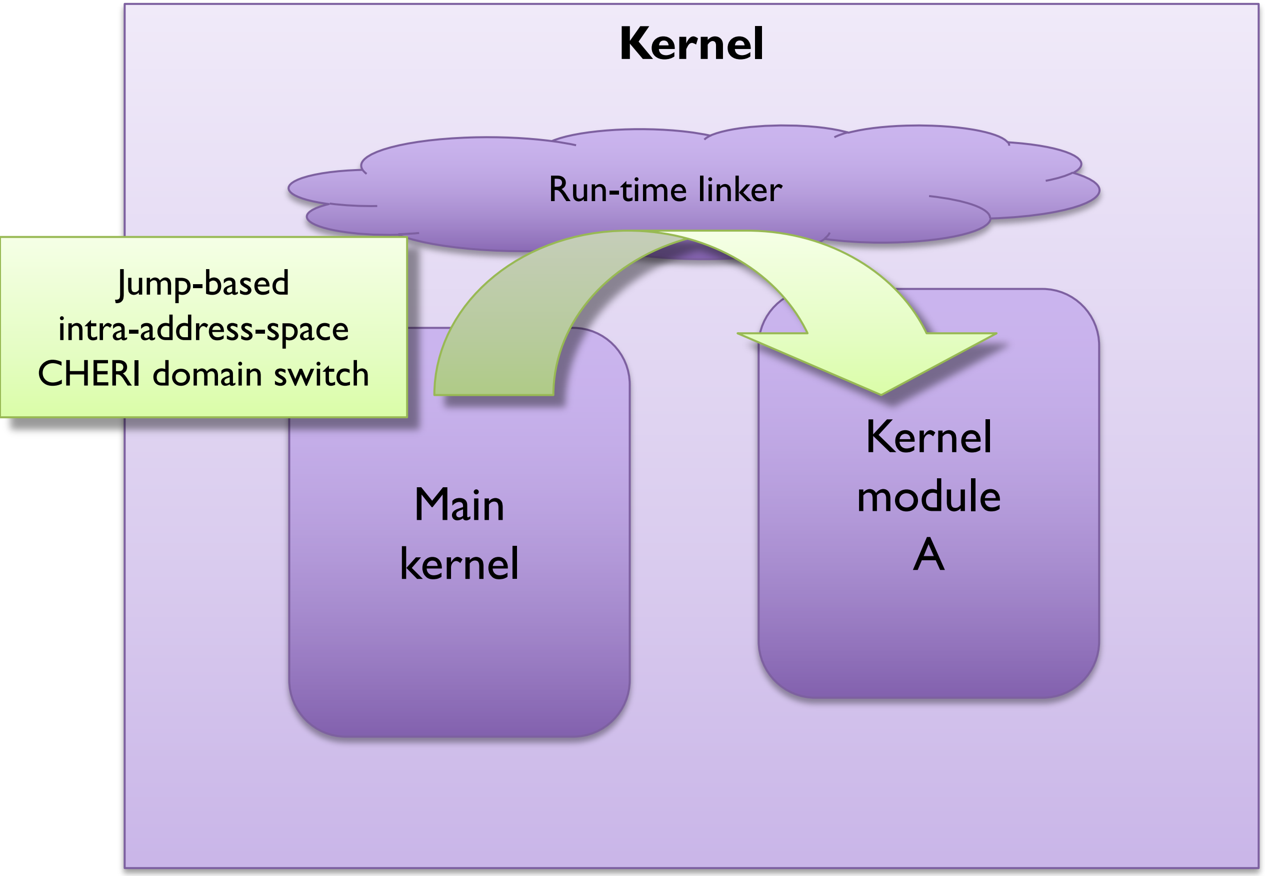
- Library compartmentalization inserts domain-transition trampolines into inter-library calls and returns
  - PLT entries are initialized with sealed trampoline capabilities that provide strong encapsulation
  - Per-target trampolines are used for branch-prediction reasons (still more tradeoffs to explore here)
  - A single "return trampoline" provides a branch-predictable reverse transition path
- Trampolines perform a number of operations relating to capability register setup/clearing, setting up return path, stack changes, etc.

Shared library compartmentalization (3/3)



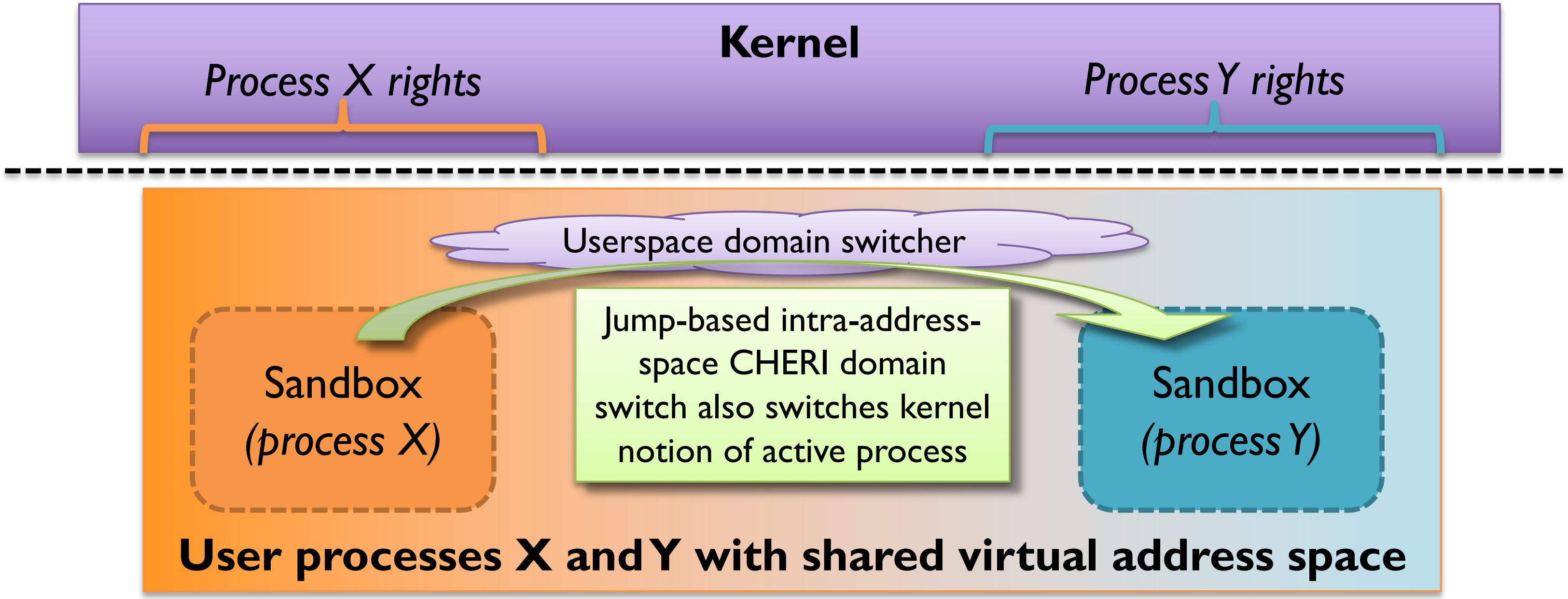
- Domain transitions on inter-library calls + returns
  - Inter-domain frames protect control flow between domains
- Stack temporal safety is hard, so we approximate
  - Per-thread trusted stack tracks domain transitions
  - Reentrant per-thread, library stack pools
  - CHERI sealing mechanism protects code transitions, data pointers from corruption

From shared libraries to kernel modules



- Can this userspace model work in the kernel as well? The kernel is actually:
  - Integrated main binary with kernel run-time linker
  - Collection of kernel modules implementing drivers, services, …
- The same model likely applies, with suitable adaptation to the kernel run-time environment
- We are developing an early prototype implementing this model

Example: CHERI co-process model



- CHERI isolates **multiple processes** within a single virtual address space
  - Kernel-provided trusted compartment switcher runs in userspace (actually a microkernel)
  - CHERI-based inter-process memory sharing + domain switching
  - A compartment's OS rights correspond to the owning process
- Inter-process context switches take **no architectural exceptions** and **do not enter the kernel**
- CHERI can be pitched as **improving IPC performance** while **retaining a (largely) conventional process model**

CHERI TRANSITION

Morello and CHERI-RISC-V

- We are pursing two CHERI adaptations to post-MIPS ISAs:
  - 2014 Joint with Arm, an experimental adaptation of 64-bit ARMv8-A **Arm Morello** multicore SoC, development board, etc. (**announced Oct. 2019; experimental SoC shipped 2022)**
  - 2017 An experimental adaptation of 32/64-bit RISC-V (**open-source research processors on FPGA**)
- Complete elaborations of the full hardware-software stack for each ISA:
  - All aspects of the architectures (e.g., ARMv8-A VM features, etc.)
  - Formal models + proofs, hardware implementations, compilers, OSes
- Potential for transition through both paths

CHERI target architectures

Architecture	Features	CHERI challenges
64-bit MIPS	1990s RISC architecture (CHERI baseline)	Our legacy research architecture. Poor code density and addressing modes: harder to differentiate 'essential' CHERI costs; few transition opportunities with MIPS
64-bit ARMv8-A	Mature and widely deployed load-store architecture	Feature-rich; exception-adverse; rich address modes; constrained opcode space; hardware page tables; virtualization features; ecosystem
32-bit and 64-bit RISC-V	Open RISC ISA in active development (MIPS + 10 years?)	Limited addressing modes (expects micro-op fusion); hardware page tables; only partially standardized; features missing (e.g., hypervisor); immature software stack

What's the smallest variety of CHERI?

Microsoft Security Response Center

Report an issu

What's the smallest variety of CHERI?

Security Research & Defense / By Saar Amar / September 6, 2022

The Portmeirion project is a collaboration between Microsoft Research Cambridge, Microsoft Security Response Center, and Azure Silicon Engineering & Solutions. Over the past year, we have been exploring how to scale the key ideas from CHERI down to tiny cores on the scale of the cheapest microcontrollers. These cores are very different from the desktop and server-class processors that have been the focus of the Morello project.

Microcontrollers are still typically in-order systems with short pipelines and tens to hundreds of kilobytes of local SRAM. In contrast, systems such as Morello have wide and deep pipelines, perform out-of-order execution, and have gigabytes to terabytes of DRAM hidden behind layers of caches and a memory management unit with multiple levels of page tables. There are billions of microcontrollers in the world and they are increasingly likely to be connected to the Internet. The lack of virtual memory means that they typically don't have any kind of process-like abstraction and so run unsafe languages in a single privilege domain.

This project has now reached the stage where we have a working RTOS running existing C/C++ components in compartments. We will be open sourcing the software stack over the coming months and are working to verify a production-quality implementation of our proposed ISA extension based on the lowRISC project's Ibex core, which we intend to contribute back upstream.

- Production-quality CHERI-RISC-V-extended Ibex core
  - Small-scale microcontroller used in OpenTitan and other use cases
  - Clean-slate memory-safe, compartmentalized OS
  - Will be open-source hardware and software
  - CHERI-RISC-V tuned for small microcontrollers
  - RISC-V embedded standardization candidate
- Collaboration across Microsoft Research, MSRC, Azure Silicon, and Azure Edge + Platform

https://msrc-blog.microsoft.com/2022/09/06/whats-the-smallest-variety-of-cheri/

SRI International

UNIVERSITY OF CAMBRIDGE

RISC-V CHERI Special Interest Group (SIG)

- Created in early October 2022, SIG acting chair is Alex Richardson (Google)
- Preparing to create first standardization task group pursuing:
  - 64-bit CHERI-RISC-V building on SRI/Cambridge's ISA
- Once IP issues are resolved, can proceed with second task group:
  - Microcontroller CHERI building on Microsoft's recent work
- Significant ISA refinement and need for high-quality reference implementation of higher-end 64-bit design

CHERI-x86 seedling

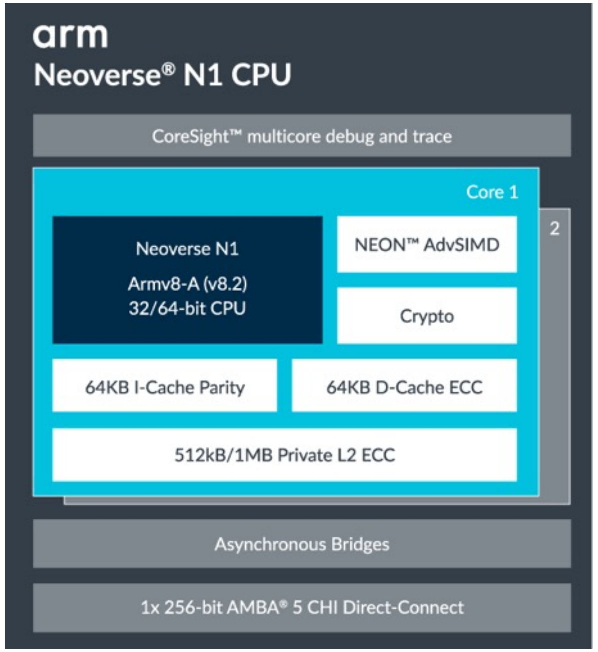
- Explore application of CHERI to the widely used x86 architecture
  - Initial prototype ISA developed and formally modeled
  - Focused on compiler targeted (“userlevel”) instruction set
  - Automatically generated test suite from formal model to enable potential future simulator and hardware implementation
  - Early low-level toolchain support; compiler support now beginning
- Proof-of-concept prototype allows design-space exploration prior to industrial engagement

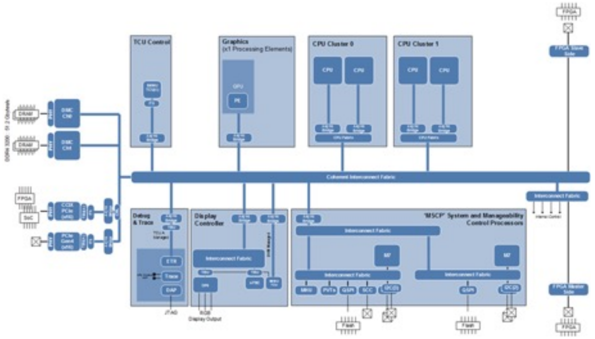
CHERI-ARM research since 2014

- Since 2014, in collaboration with Arm, we have been pursuing joint research to experimentally incorporate CHERI into ARMv8-A:
  - Develop CHERI as an architecture-neutral and portable protection model implemented in multiple concrete architectures
  - Refine and extend the CHERI architecture – e.g., capability compression, tagging µarch, domain transition, and temporal safety
  - Apply concept of architecture neutrality to the CHERI-enabled software stack, including compiler, OS, and applications
  - Expand software: large-scale application experiments, OS use, debuggers, …
  - Extend work in formal modeling and proofs to an industrial-scale architecture
- Solve arising practical {hardware, software, …} problems as part of the research
- Build evidence, demonstrations, SW templates to support potential CHERI adoption

ISCF: Digital Security by Design (UKRI)

- 5-year **Digital Security by Design** UKRI program: £70M UK gov. funding, £117M UK industrial match, to create CHERI-ARM demonstrator SoC + board with proven ISA
- Leap supply-chain gap that makes adopting new architecture difficult – in particular, validation of concepts in microarchitecture, architecture, and software “at scale”
- Support industrial and academic R&D (EPSRC, ESRC, InnovateUK)
- Baseline CPU is Neoverse N1; reuses existing SoC/board designs
- Collaborative review distillation of CHERI ISAv8; experimental additions relating to temporal safety, compartmentalization
- Science designed allowed: Multiple architectural + microarchitectural design choices for software-based evaluation
- 2020 emulation models; **2022 Morello board shipped!**





UNIVERSITY OF CAMBRIDGE

Digital Security by Design

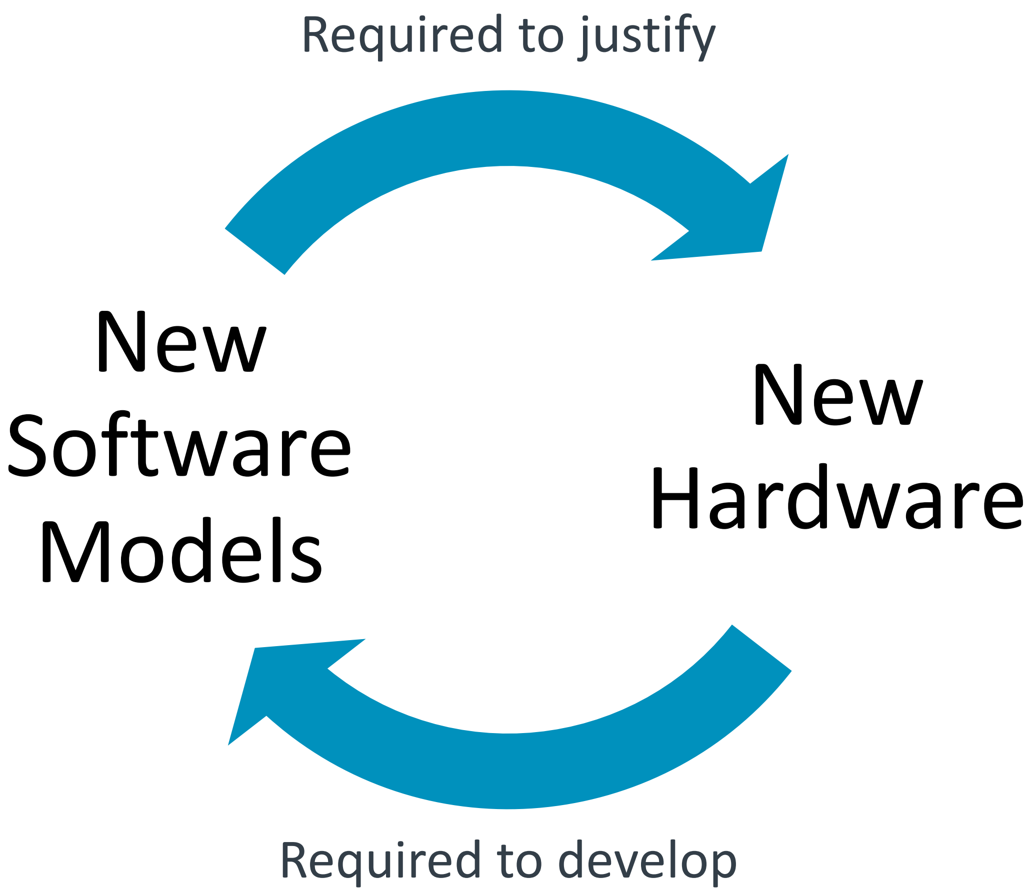
Richard Grisenthwaite

SVP Chief Architect and Fellow

Richard.Grisenthwaite@arm.com

Challenges with creating substantially new architecture

INDUSTRIAL STRATEGY
UK Research and Innovation



arm

Why is Arm interested in the CHERI architecture

INDUSTRIAL STRATEGY
UK Research and Innovation

- Arm had been working with UoCambridge on CHERI for some 4-5 years
- Big step to addressing security based on strong fundamental principles
- Addresses spatial memory safety robustly and some ideas for temporal safety
  - Memory safety issues reported to be involved with ~70% of vulnerabilities (Matt Miller, BlueHat IL, 2019)
- Has scope to be the foundation of a new mechanism for compartmentalisation
  - Potentially far cheaper than using translation tables
- Interesting scope to address temporal safety issues as well as spatial ones….
- Many of the Arm software vendors are similarly interested in the possibilities of CHERI
  - Microsoft, Google and others have expressed strong interest in exploring the concept…
  - … but lots of questions about the real-world performance costs and usage models
  - …understanding the intended usage models is important to refine the architectural features
- But is a novel thing to do with additional costs to the system and software
  - Adding a 129th tag bit has a lot of impacts to the memory system
  - it is an ABI change, so non-trivial costs for compatibility for some uses

arm

IP Position

- Today’s CPU architectures have largely the same basic functionality
  - “Similar but different” approaches to most aspects of system architecture
  - Small scale optimisations exist
- This position very beneficial for the porting of system software
  - Anything that fundamentally changes the system software architecture is likely to be ignored
- Arm believes that this reality needs to continue with capabilities
  - Implication is that we’d like the world’s leading architectures to adopt capabilities
  - The Digital Security by Design program

Arm Morello specification

INDUSTRIAL STRATEGY
UK Research and Innovation

Arm® Architecture Reference Manual Supplement Morello for A-profile Architecture

Document number	DDI0606
Document version	A.f
Document confidentiality	Non-confidential

Important message

Morello is a prototype architecture, which has a particular meaning to Arm of which the recipient must be aware as follows:
Subject to change without consent of all parties, and it is not committed for product development.
Includes the majority of expected features.
Includes detail on the majority of expected features.
Includes some necessary information from documentation relating to earlier architectures, but some cross-referencing might be necessary.
See the architecture release notes for more detail.
No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.

- Experimental application of CHERI ISAv8 to ARMv8-A
- Much richer base ISA .. Much longer spec - 2,155 pages excluding additional material!
- Describes ISA as implemented in Arm Morello FVP and processor/SoC
- Includes recent features such as sentry and load-side barrier support

arm

The Morello Board

- An Industrial Demonstrator of a Capability architecture
- Uses a prototype capability extension to the Arm Architecture
  - Prototype is a “superset” of what could be adopted into the Arm architecture
- Use of a superset of the architecture is very unusual
  - Also unrealistic as a commercial product – there will be some frequency effects
  - However, there are tight timescales so architecture is nearly complete now
- The superset of the architecture will allow a lot of software experimentation
  - Various different mechanisms for compartmentalisation
  - Collection of features for which the justification is unclear
  - Techniques for holding the capability tag bit
- Architecture will have formally proved security properties (with UoC and UoE)
- **Morello Board will be the ONLY physical implementation of this prototype architecture**
  - **Learnings from these experiments will be adopted into a mainstream extension to the Arm architecture**
  - **NO COMMITMENT TO FULL BINARY COMPATIBILITY TO THE PROTOTYPE ARCHITECTURE**
    - **But successful concepts are expected to be carried forward into the architecture and can be reused there**

INDUSTRIAL STRATEGY
UK Research and Innovation

arm

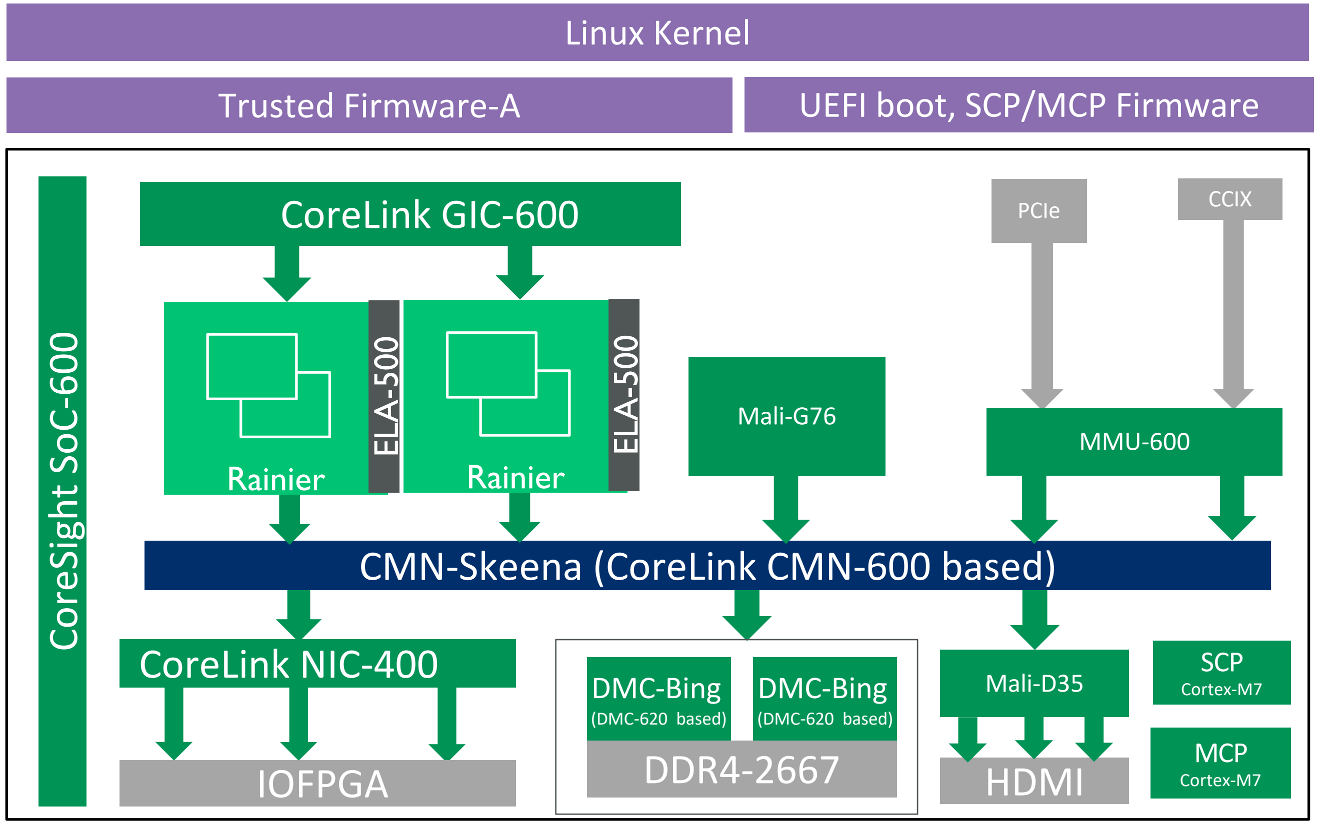
Morello Board overview (subject to change)

- Quad core bespoke high-end CPU with prototype capability extensions
  - Backwards compatibility with v8.2 AArch64-only
  - Based on Neoverse N1 core
    - Multi-issue out-of-order superscalar core with 3 levels of cache
  - Build in 7nm process
  - Targeting clock frequency around 2GHz
- Reasonable performance GPU and Display controller
  - Standard Mali architecture core – not extended with capability
  - Supports Android
- PCIe and CCIx interfaces including to FPGA based accelerators
- FPGA for peripheral expansion
- SBSA compliant system
- 16GB of System Memory (expandable to 32GB – tbc)

Morello Board: Capability Hardware Prototype Platform

INDUSTRIAL STRATEGY
UK Research and Innovation

- Silicon implementation of a Capability Hardware CPU Instruction Set Architecture
  - Implements Morello Profile for A-class Prototype Architecture
  - Two clusters each of two Rainier CPUs
  - Interconnect and Memory Controller support for tagged memory
  - Two channel DDR4 DRAM interface
  - PCIe Gen3 and Gen4 x16 interface
  - CCIX (Cache Coherent Interconnect for Accelerators) interface
  - Mid-range GPU, display processor and HDMI output
  - On standard uATX form factor board



Supporting Arm system IP: GIC-600 (Generic Interrupt Controller), MMU-600 (IO MMU), Dynamic Memory Controller derived from DMC-620, SoC-600 (SoC Debug and Trace), Coherent Mesh Network derived from CMN-600, NIC-400 (Non-coherent interconnect)

Supporting 3rd party system IP/hardware: PCIe/CCIX Root Complex (PHY and controller), DDR4/3 PHY, DDR4 memory, IO FPGA

Open-source software stack

arm

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

Technical Report UCAM-CL-TR-982
ISSN 1476-2986

Number 982

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Arm Morello Programme:
Architectural security goals
and known limitations

Robert N. M. Watson, Graeme Barnes,
Jessica Clarke, Richard Grisenthwaite,
Peter Sewell, Simon W. Moore,
Jonathan Woodruff

July 2023

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

- Framing security direction and disclaimers:
  - Architectural security aims and experimental validation
  - Constraints of the Armv8.2-A baseline ISA
  - Limitations of the experimental software stack
  - Limitations on the hardware threat model
- Important to understand what Morello can do – and cannot; e.g.,
  - Has enabled 50+MLoC CHERI C/C++ code corpus
  - No expectation to resist Spectre or Rowhammer

https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-982.pdf

SRI International · UNIVERSITY OF CAMBRIDGE

Early performance results from the prototype Morello microarchitecture (September 2023)

Technical Report

UCAM-CL-TR-986
ISSN 1476-2986

Number 986

UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Early performance results from the prototype Morello microarchitecture

Robert N. M. Watson, Jessica Clarke, Peter Sewell, Jonathan Woodruff, Simon W. Moore, Graeme Barnes, Richard Grisenthwaite, Kathryn Stacer, Silviu Baranga, Alexander Richardson

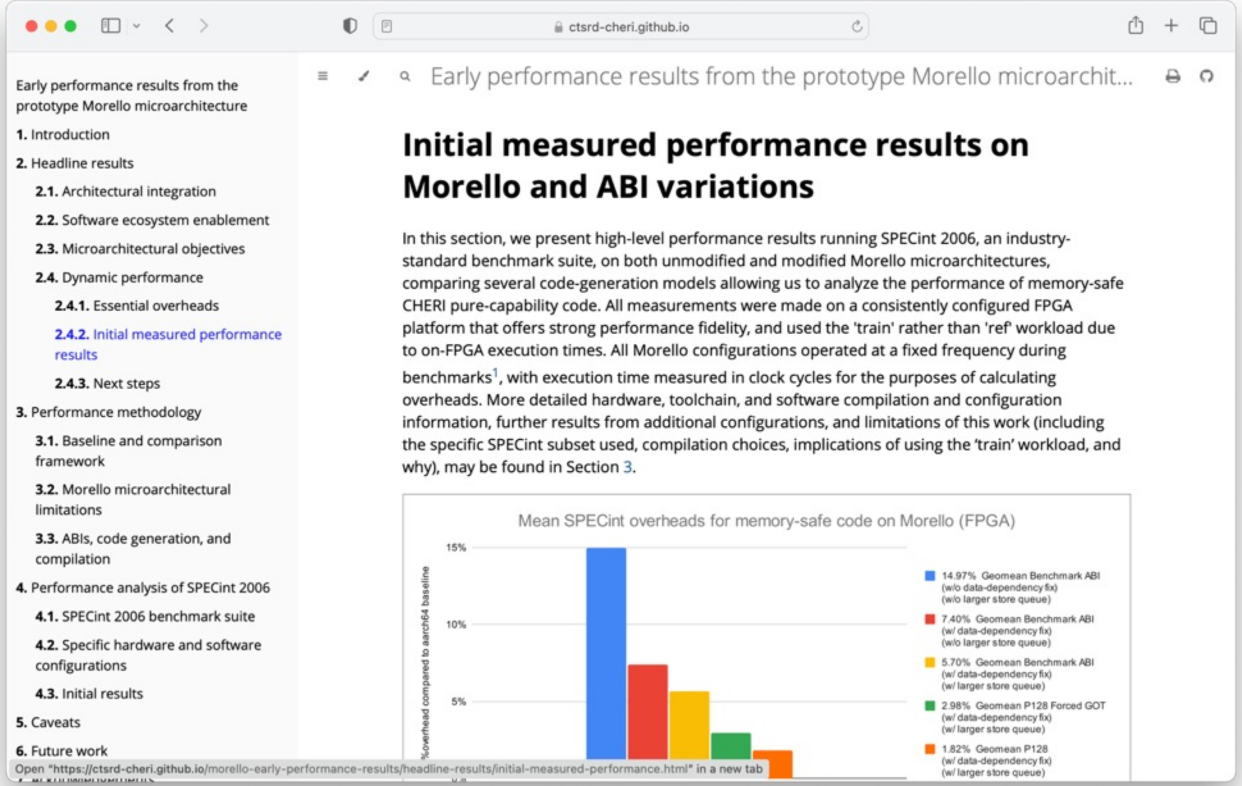
September 2023

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

- Performance analysis of SPECint 2006 on Morello
  - Reminder: Morello is prototype architecture and microarchitecture; no production optimization cycle possible on DSbD timeline
  - Baseline Morello microarchitecture "as shipped"
  - Modified Morello designs on FPGA addressing discovered limitations / re-tuning parameters
- "Benchmark ABI" and "P128" code models to improve predictions for future mature microarchitecture
- Best available spatial safety overhead on Morello prototype microarchitecture, with refinements, for SPECint 2006: **5.7%**
- Worst projected spatial safety overhead on anticipated mature microarchitecture for SPECint 2006: 1.8% - 3.0%

https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-986.pdf

Early performance results from the prototype Morello microarchitecture (live website)



- Live version of the website will be updated as understanding improves
- Currently in sync with TR, but will see further updates in coming months (see Version History)
  - Looking at topics such as the impact of dynamic linking
- Complements Benchmarking Guidance section in **Getting Started with CheriBSD**

https://ctsrd-cheri.github.io/morello-early-performance-results/cover/index.html

UK EPSRC DSbD research program 2020-2023

EPSRC Competition

- £10M Research funding
  - £7M from ISCF/DSbD
  - £3m from DCMS
- The EPSRC call covered 3 areas:
  - Capability enabled hardware proof and software verification
  - Impact on system software and libraries
  - Future implications of capability enabled Hardware
- Projects starting July-Oct

Selected Projects

AppControl: Enforcing Application Behaviour through Type-Based Constraints
Dr Wim Vanderbauwhede (University of Glasgow)

CapableVMs – Capable Virtual Machines
Dr Laurence Tratt (King's College London) & Dr Jeremy Singer (University of Glasgow)

CAPcelerate: Capabilities for Heterogeneous Accelerators
Dr Timothy Jones (University of Cambridge)

CapC: Capability C semantics, tools and reasoning
Dr Mark Batty (University of Kent)

CAP-TEE: Capability Architectures for Trusted Execution
Dr David Oswald (University of Birmingham)

CHaOS: CHERI for Hypervisors and Operating Systems
Dr Robert Watson (University of Cambridge)

CloudCAP: Capability-based Isolation for Cloud-Native Applications
Prof Peter Pietzuch (Imperial College London)

HD-Sec: Holistic Design of Secure Systems on Capability Hardware
Professor Michael Butler (University of Southampton)

SCorCH: Secure Code for Capability Hardware
Dr Giles Reger (The University of Manchester)
Prof Daniel Kroening (University of Oxford)

Department for Digital, Culture Media & Sport

EPSRC
Engineering and Physical Sciences Research Council

- 9 EPSRC projects funded across 10 UK universities
- Several InnovateUK industrial projects supporting exploration, evaluation, demonstration

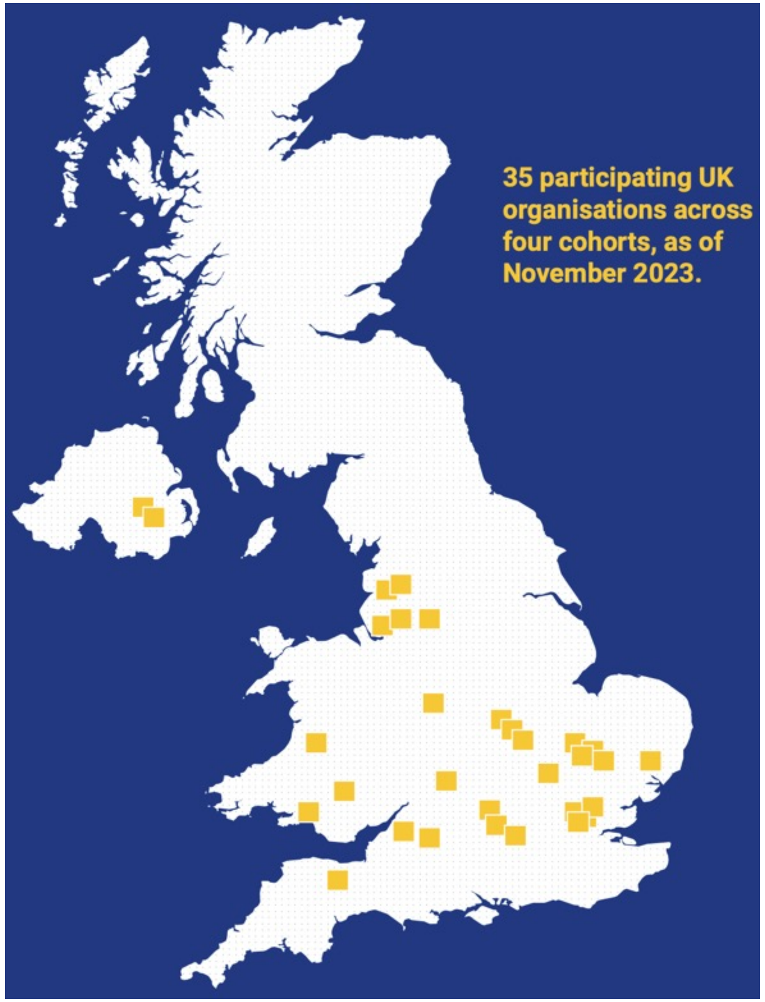
SRI International

UNIVERSITY OF CAMBRIDGE

DSbD Technology Access Programme

- Digital Security by Design (DSbD) runs the Technology Access Programme (TAP) for UK-based companies to experiment with CHERI and Morello
- We have collaborated with ~35 companies that have been porting their products or prototyping new projects on Morello boards
- Several of these companies reported that, using Morello, they found vulnerabilities in their code and analysed past vulnerabilities against CHERI

DSbD TAP Cohort 1-4



Programme scale so far:

- +15 million lines of code ported to Morello by Cohorts 1, 2, 3 and 4
- 32 networking and learning events
- Multi-sector and cross-discipline involvement

Source: Digital Catapult, DSbD TAP Showcase booklet

DSbD TAP Cohort 1-4: example projects

- Cohort 1, RealVNC: Memory-safe remote access VNC client and server
- Cohort 2, CAN-PHANTOM: Memory-safe CAN-based vehicle immobiliser based on libusb from CheriBSD
- Cohort 3, JET Connectivity: Memory-safe 5g-enabled base station
- Cohort 4, rtegrity: Memory-safe and compartmentalised user-space storage stack based on SPDK and DPDK

CHERI REFERENCE SOFTWARE STACK

Why port the CHERI stack to Morello?

- **Validate** the Morello architecture (functional, sufficient)
- **Evaluate** the Morello implementation (performance, energy use, …)
- **Provide reference software semantics** (spatial and temporal safety, compartmentalization, POSIX integration, OS kernel use, …) that will be applicable to other adaptations
- **Act as a template and prototyping platform** for at-scale industrial and academic demonstration, including providing adaptations of common software dependencies (e.g., widely used libraries)
- **Provide a platform for future software research,** asking questions about what we can use CHERI for in {operating systems, compilers, language runtimes, applications, …}
- **Enable a growing academic and industrial community** around CHERI and Morello, including dozens of UK universities and companies associated with DSbD

Caution: Research software!

- **The baseline compiler toolchain and OS stack are themselves research**
  - This means unknown risks, hard-to-predict schedules, and inevitable direction changes
- **Application Binary Interface (ABI) stability**
  - ABIs are a key research area; there are 2x Morello ABIs, and there will be [many?] more
  - This limits long-term binary compatibility guarantees for compiled software (for example)
- **Software performance optimization with a limited corpus**
  - Right now, we're just happy things are working, but we will get beyond that soon!
- **Supporting a large and diverse audience of consumers with different objectives**
  - Engineering constraints limit objectives and support (e.g., software updates)
- **Software adaptation workload**
  - Some code ports trivially (e.g., Qt/KDE stack) and other code doesn't (e.g., JITs)

CHERI prototype software stack on Morello

- **Complete open-source software stack** from bare metal up: compilers, toolchain, debuggers, hypervisor, OS, applications – all demonstrating CHERI
- Rich CHERI feature use, but fundamentally incremental/hybridized deployment

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

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

- FreeBSD kernel + userspace, application stack
- Kernel spatial and referential memory protection
- Userspace spatial, referential, and temporal memory protection
- Co-process compartmentalization (development branch)
- Linker-based compartmentalization
- Morello-enabled bhyve Type-2 hypervisor
- ARMv8-A 64-bit binary compatibility for legacy binaries

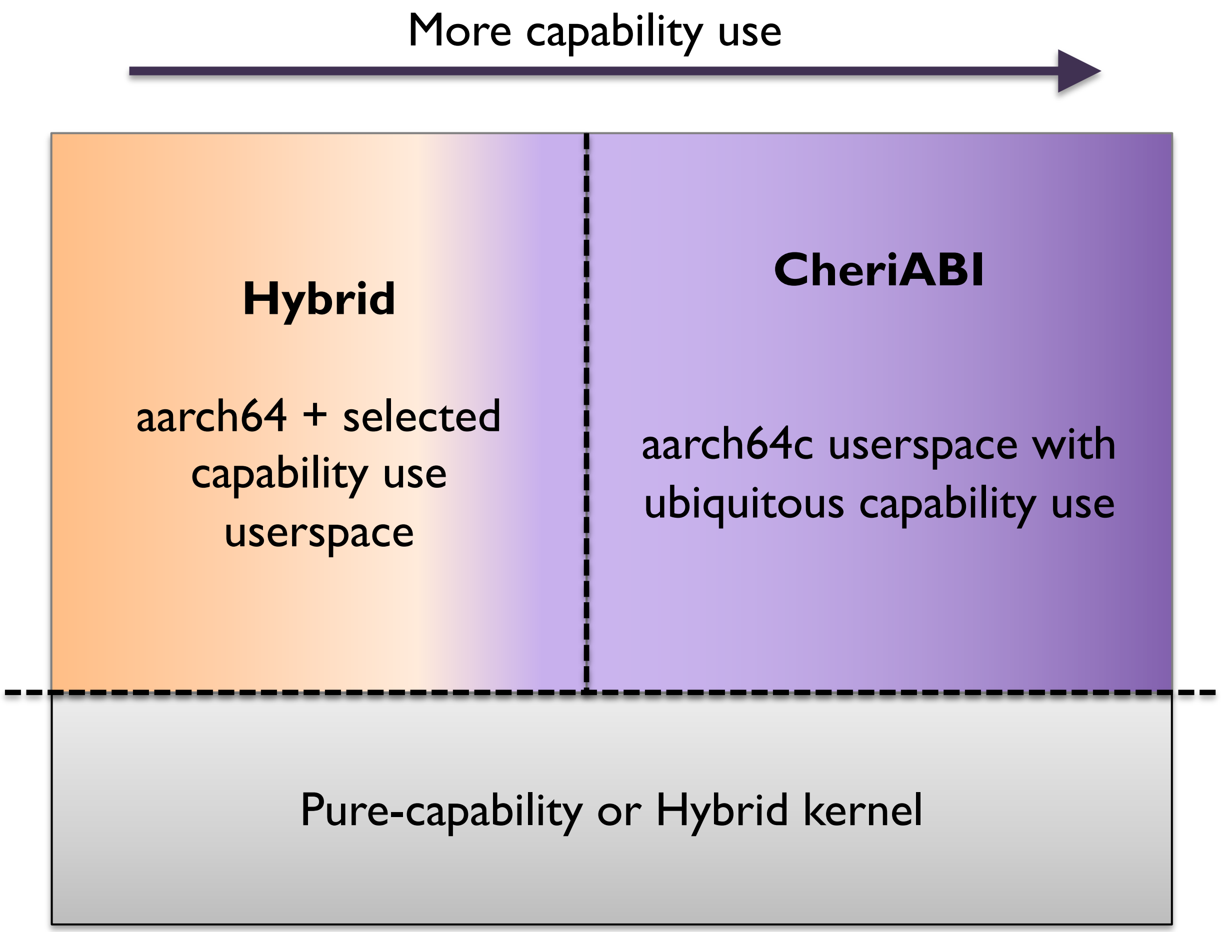
Android (Arm)
(Morello only)

Linux (Arm)
(Morello only)

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

Baseline CHERI Clang/LLVM from SRI/Cambridge; Morello adaptation by Arm + Linaro

(At least) two code generation / ABI targets



- **Hybrid code** is primarily aarch64 but with selected capability use:
  - Kernel: Mostly aarch64 with capability use for system-call arguments, context switching, virtual memory, signals
  - Userspace: Runs off-the-shelf arm64 programs without modification
- **Pure-capability code** implements all data and control-flow pointers with capabilities:
  - Kernel and userspace both spatially and referentially space
  - In the future userspace temporally safe

FreeBSD base, ports/packages

Base Base FreeBSD OS including kernel and key libraries, shells, daemons, and command-line tools

Well adapted to CHERI

Ports Build infrastructure + FreeBSD adaptation patches – roughly 30,000 mainstream open-source libraries, runtimes, and application

Early prototype

Packages Prebuilt binary packages built from ports, installed and managed using the pkg(8) package manager

Early prototype

We provide a full set of ~20K-30K aarch64 (non-CHERI) packages to run on CheriBSD/Morello to use while the CheriABI collection matures.

Maturing CHERI software artifacts

Feature	Status	Availability
3rd-party packages (Hybrid)	23K **memory-unsafe** software packages with strong functionality expectations	Since May 2022 (22.05 release)
3rd-party packages (CheriABI)	11K **memory-safe** software packages with mixed functionality expectations	Since May 2022 (22.05 release) **Up from 9k packages in 23.11**
Morello GPU device drivers	Memory-safe kernel and user drivers,	Since December 2022 (22.12 release)
Benchmark ABI support (+3rd-party packages)	Support for modified code generation addressing Morello bounds prediction	**Shipping in 23.11** (roughly the same packages as CheriABI)
Userlevel heap temporal safety	Prototype implements strong temporal safety, developed with Microsoft; testing required	**Shipping in 23.11** (pretty experimental)
Linker-based compartmentalization	Introduces strong encapsulation boundaries around UNIX libraries with no modification	Since 22.12 (very experimental); **Significant improvements in 23.11**
bhyve (Type-2) hypervisor	Prototype boots pure-capability guest OS, validation required	**Shipping in 23.11** (very experimental)
Co-process compartmentalization	Prototype runs some compartmentalized software (e.g., OpenSSL); API co-design	Planning to ship in 2024

Ease of adoption compared to high-level languages

Language	Approximate open-source LoC*	Memory safe	Memory safe with CHERI
C	10,317,800,000	✗	✓
C++	2,937,550,000	✗	✓
Java	2,600,000,000	✓	✓
Rust	39,500,000	✓	✓

In the past 6 months, the CHERI project has converted more lines of open-source code to memory safety than the Rust project has created in its entire history.

* Synopsys Black Duck Open Hub: https://www.openhub.net/languages

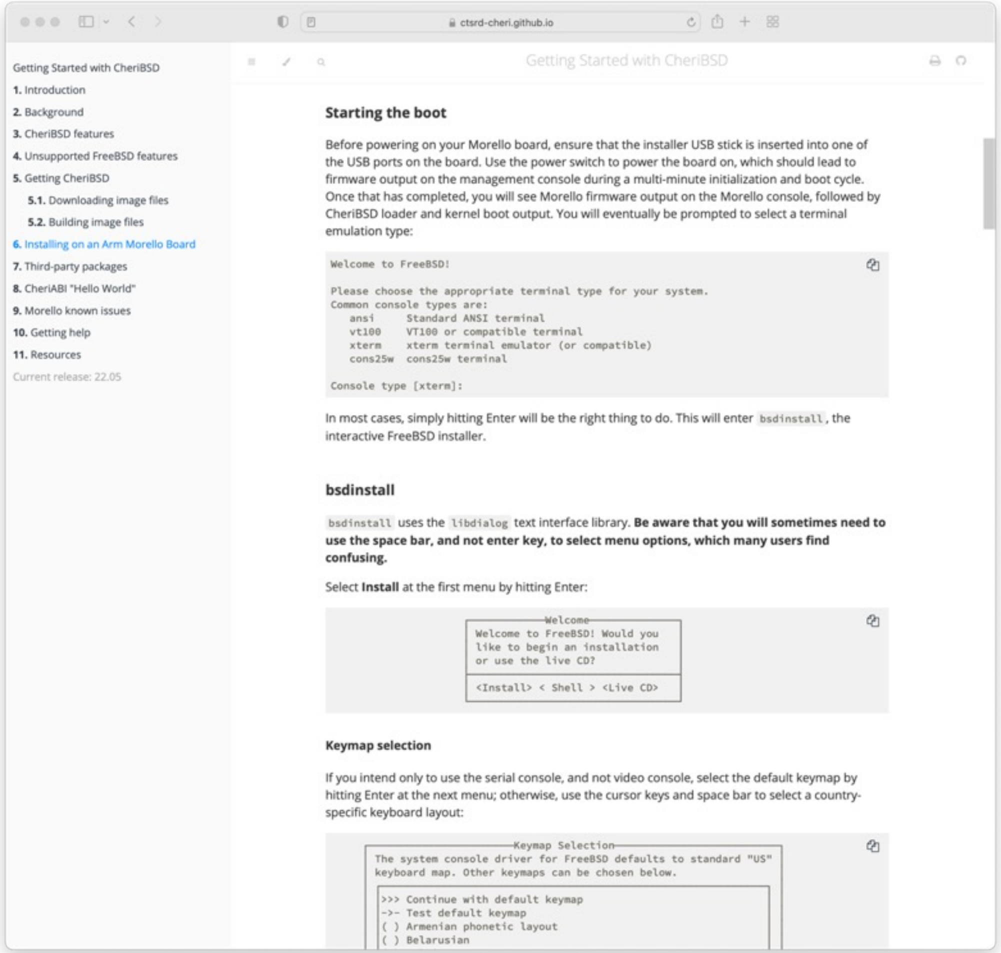
SRI International

UNIVERSITY OF CAMBRIDGE

Could we achieve practical memory safety* for C/C++ desktop/server/embedded stacks within 4 years without a total software 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.

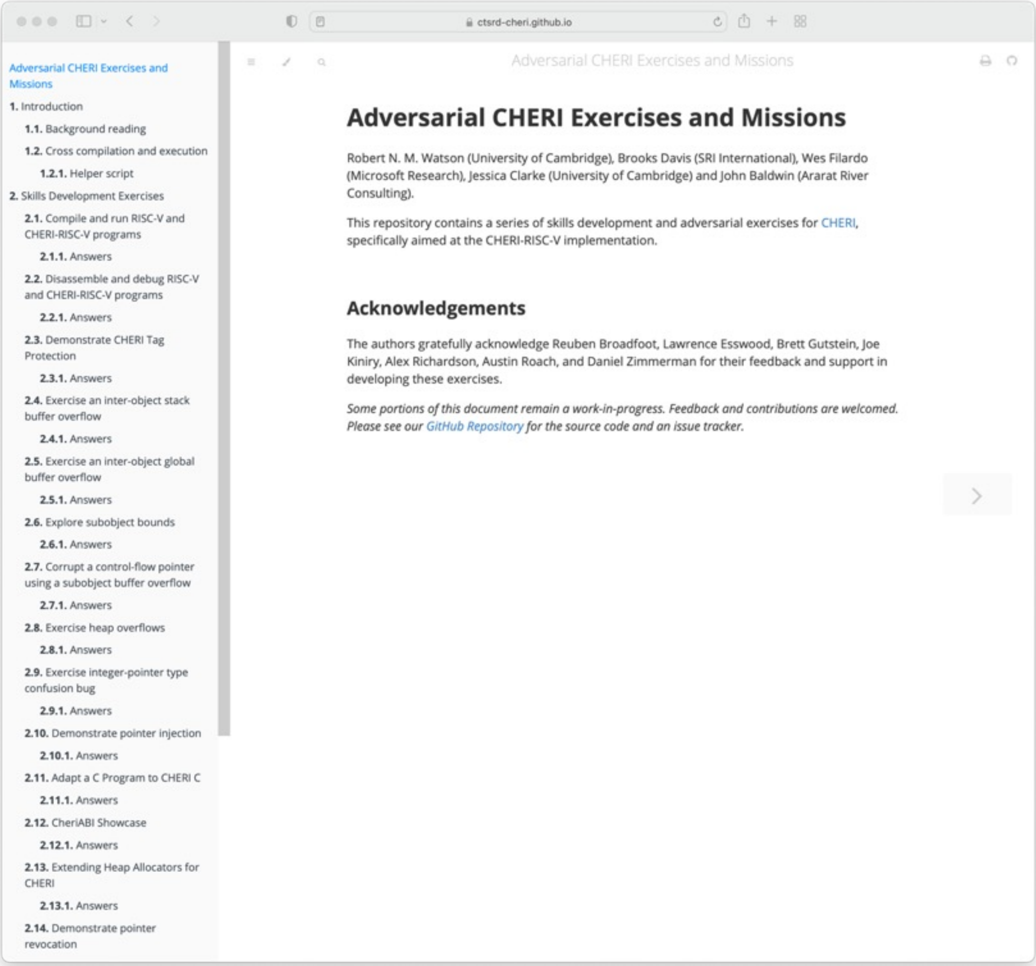
Getting Started with CheriBSD



- Introduces CheriBSD
- Steps you through installation on a Morello board using a USB stick image that you can download
- Describes third-party package system and pkg64/pkg64c
- Illustrates "hello world" compilation and debugging
- Describes some known issues
- Explains how to get support

https://ctsrd-cheri.github.io/cheribsd-getting-started/

Adversarial CHERI Exercises and Missions



- CHERI training exercises for developers, red teams, and bug bounties
- Adversarial missions where we want to understand exploitation better
- CHERI software adaptation
- Assume a strong level of knowledge about C, code generation, exploitation
  - (E.g., GOTs, PLTs, ROP, and JOP)
- Targets Morello and CHERI-RISC-V

https://ctsrd-cheri.github.io/cheri-exercises/

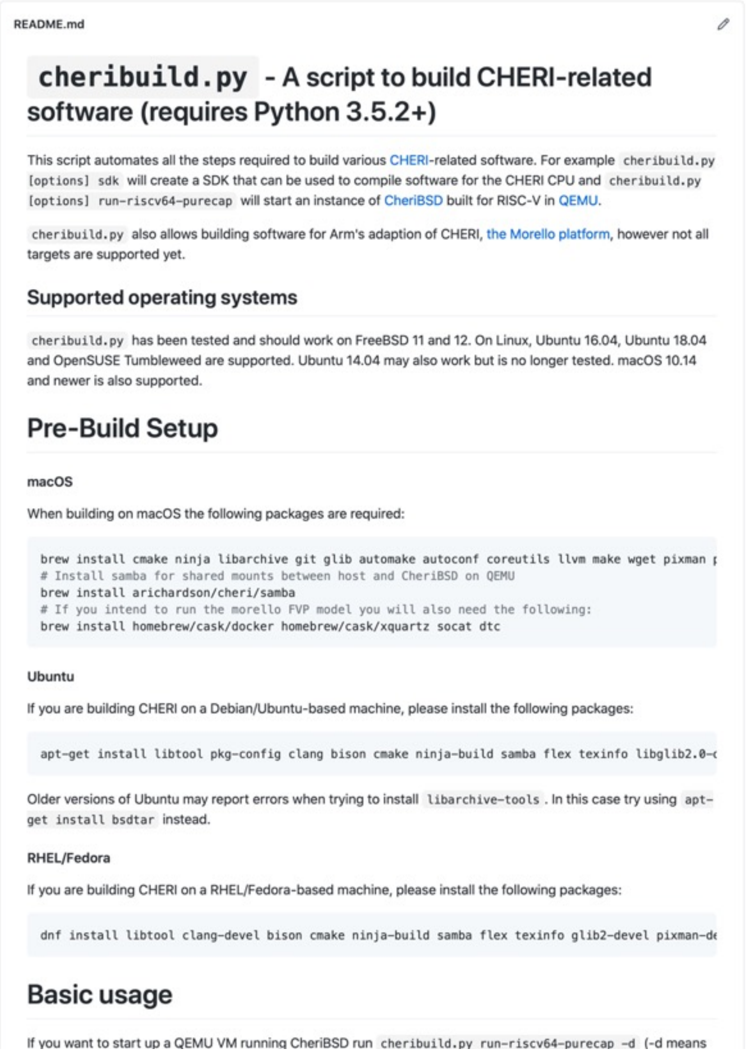
SRI International

UNIVERSITY OF CAMBRIDGE

CHERI software stack support channels

- cheri-cpu.slack.com Slack
  - Visit the CHERI website to request an invitation email/link
- Forthcoming mailing lists (not yet live)
  - cl-cheribsd-announce — Low-traffic announcement
  - cl-cheribsd-discuss — General discussion and support
  - cl-cheribsd-security — Report security issues
- Sundry issue trackers in the github.com/CTSRD-CHERI organization
- Not just "How do I get the software to work", but also to assist with **experimental design**, **interpreting results**, and **seeking improvements**

How to obtain and install the CHERI software stack


README.md

`cheribuild.py` - A script to build CHERI-related software (requires Python 3.5.2+)

This script automates all the steps required to build various CHERI-related software. For example `cheribuild.py [options] sdk` will create a SDK that can be used to compile software for the CHERI CPU and `cheribuild.py [options] run-riscv64-purecap` will start an instance of CheriBSD built for RISC-V in QEMU.

`cheribuild.py` also allows building software for Arm's adaption of CHERI, the Morello platform, however not all targets are supported yet.

Supported operating systems

`cheribuild.py` has been tested and should work on FreeBSD 11 and 12. On Linux, Ubuntu 16.04, Ubuntu 18.04 and OpenSUSE Tumbleweed are supported. Ubuntu 14.04 may also work but is no longer tested. macOS 10.14 and newer is also supported.

Pre-Build Setup

macOS

When building on macOS the following packages are required:

```
brew install cmake ninja libarchive git glib automake autoconf coreutils llvm make wget pixman p
# Install samba for shared mounts between host and CheriBSD on QEMU
brew install arichardson/cheri/samba
# If you intend to run the morello FVP model you will also need the following:
brew install homebrew/cask/docker homebrew/cask/xquartz socat dtc
```

Ubuntu

If you are building CHERI on a Debian/Ubuntu-based machine, please install the following packages:

```
apt-get install libtool pkg-config clang bison cmake ninja-build samba flex texinfo libglib2.0-c
```

Older versions of Ubuntu may report errors when trying to install `libarchive-tools`. In this case try using `apt-get install bsdtar` instead.

RHEL/Fedora

If you are building CHERI on a RHEL/Fedora-based machine, please install the following packages:

```
dnf install libtool clang-devel bison cmake ninja-build samba flex texinfo glib2-devel pixman-de
```

Basic usage

If you want to start up a QEMU VM running CheriBSD run `cheribuild.py run-riscv64-purecap -d` (-d means

- One build tool to rule them all: cheribuild

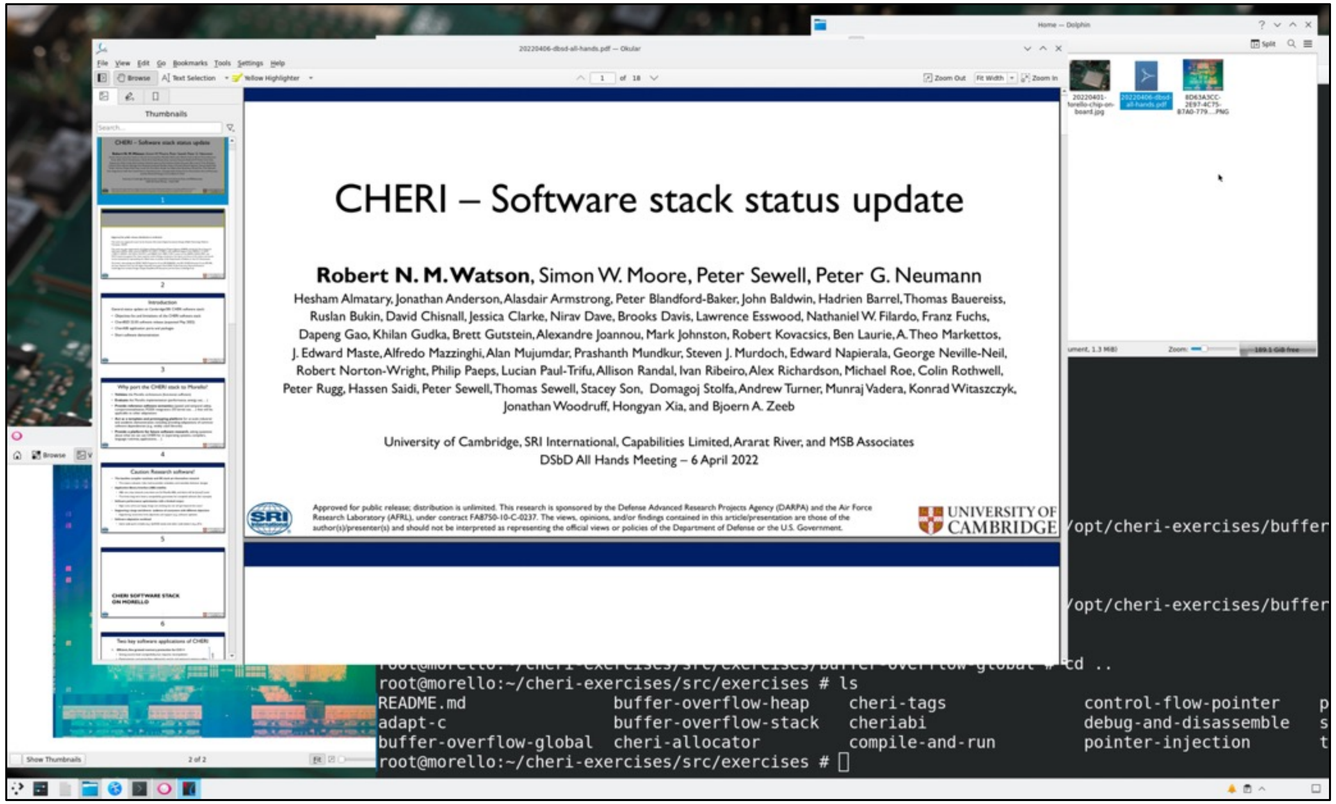
  https://github.com/CTSRD-CHERI/cheribuild

- Builds, installs, and/or runs:
  - QEMU CHERI-RISC-V and Morello, Morello FVP
  - CheriBSD/CHERI-RISC-V and Morello disk images
  - Small suite of adapted third-party applications
- Up and running with one command (CHERI-RISC-V):

  ./cheribuild.py --include-dependencies run-riscv64-purecap

CHERI/MORELLO DESKTOP STUDY

2021 desktop pilot study results



Developed:

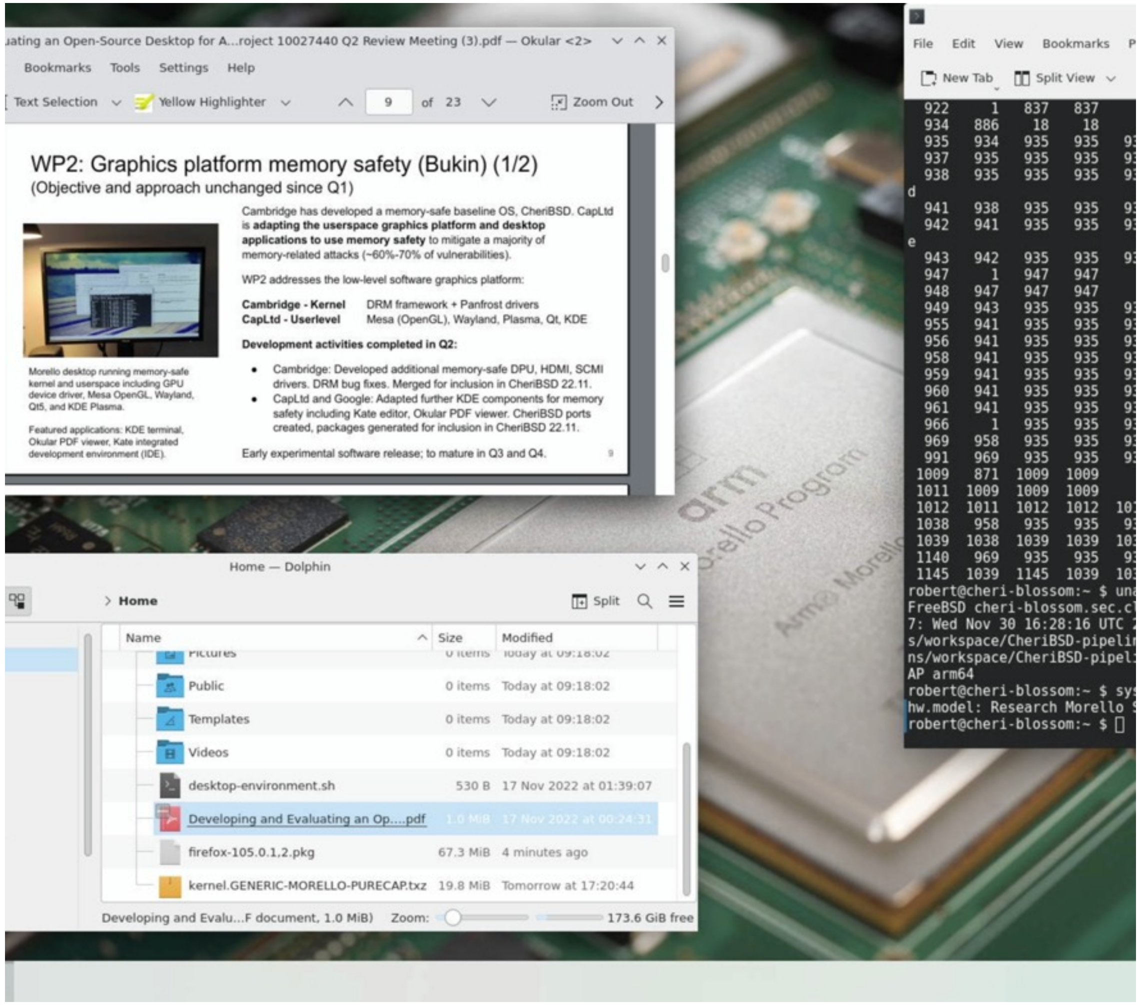
- **6 million lines of C/C++ code** compiled for memory safety; modest dynamic testing
- **Three compartmentalization whiteboard case studies** in Qt/KDE

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 …

2022.12 Morello memory-safe desktop software stack



Roughly 30MLoC on a shipping Arm Morello board today, with memory-safe:

- CheriBSD kernel with DRM + Panfrost drivers
- CheriBSD userspace with libraries, OpenSSH, ...
- OpenGL, Wayland display server
- Plasma, KDE base applications including Dolphin, Okular, Konsole.

Also shipping in December 2022 with:

- Aarch64 CHERI/Morello-aware GDB debugger
- 9K CheriABI packages, 20K aarch64 ("legacy") packages; notable exclusions for language runtimes

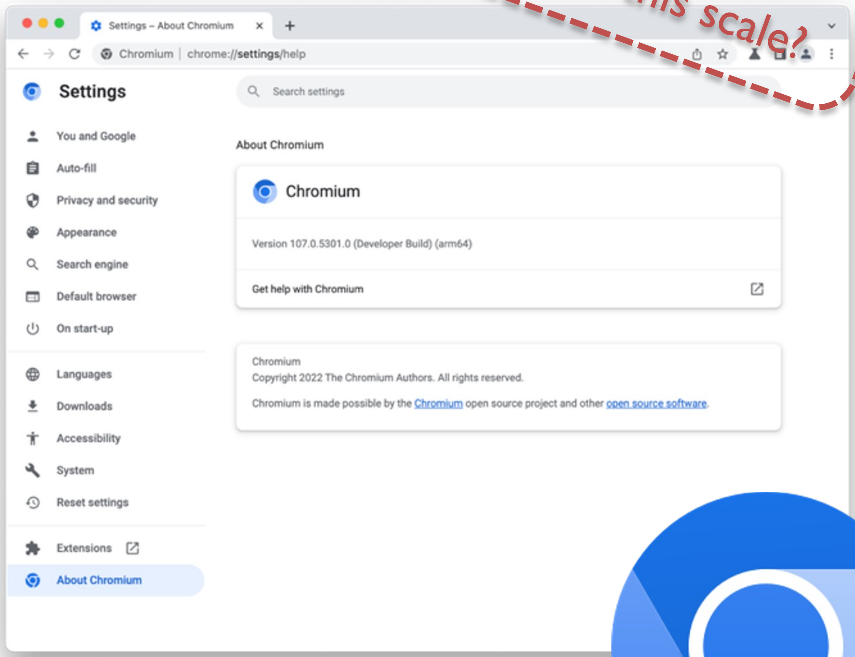
Now on to the grand challenges

- We are now within reach of an exciting – and historically highly vulnerable – application corpus to which we can apply CHERI protections
- Memory-safe desktop applications at scale – especially those that contain one or more language runtimes:
  - Web browsers
  - Mail readers
  - Office suites
- Extending this to fine-grained compartmentalization as software prototypes mature – library compartmentalization, coprocesses, further models, …
- For example: UKRI- and Google-funded efforts around the Chromium web browser at CapLtd, Kings College London, Arm, and Cambridge

Memory Safety Grand challenge: Google Chromium

- "The real thing":
  - Foundation for Google Chrome, Microsoft Edge, Microsoft Teams, Electron, …
  - Over 35MLoC, >190 library dependencies
  - V8, an intimidatingly real language runtime
  - Code from numerous diverse origins and in countless forms of idiomatic C and C++
  - Vast wealth of past vulnerabilities to use in evaluation
  - Performance critical components
  - Memory-safety and compartmentalization objectives
- ~9 staff months so far, most effort went into V8 adaptation
  - **V8 now running test suite with complete JIT support**
- Pilot project supported by UKRI, and Google

How could we compartmentalize software at this scale?



CONCLUSION

Some potential software research areas

- **Clean-slate OSes and languages**

  Current research has focused on incremental CHERI adoption within current software and languages. How would we design new OSes, languages, etc., assuming CHERI as an ISA baseline?

- **Compilers, language runtimes, and JITs**

  How can we mitigate the performance overheads of more pointer-dense executions, such as with language runtimes? Are vulnerabilities in code generated by compilers and JIT susceptible to mitigation using CHERI? How does CHERI break or potentially improve current compiler analyses and optimization?

- **Further C/C++ protections with CHERI**

  We have focused on spatial, referential, and temporal memory safety for C/C++. But the CHERI primitives could assist with data-oriented protections, garbage collection, type checking, etc. Could these improve security, and at what performance cost?

- **Safe and managed languages**

  Languages such as Java, Rust, C#, OCaml, etc., offer strong safety properties, but frequently depend on C/C++ runtimes and FFI-linked native code. Can CHERI provide stronger foundations for higher-level language stacks?

- **Virtualization**

  Can memory protection usefully harden hypervisors? Can we compartmentalize hypervisors? Can CHERI offer a better mechanism for virtualizing code than an MMU?

- **Debuggers and tracing**

  Debugging/tracing tools rely on high levels of privilege to operate. How can we reduce their privilege to mitigate vulnerabilities in these tools? With stronger architectural semantics, is new dynamic analysis possible?

- **Software compartmentalization tools**

  Granular software compartmentalization offers vulnerability mitigation through privilege reduction and strong encapsulation. How should current applications be refactored, and new applications be designed, to accomplish maintainable and more secure software?

- **Security evaluation and adversarial research**

  What is the impact of CHERI on known vulnerabilities and attack techniques? How does a CHERI-aware attacker change their behavior? Could formal models and proofs support stronger security arguments for CHERI?

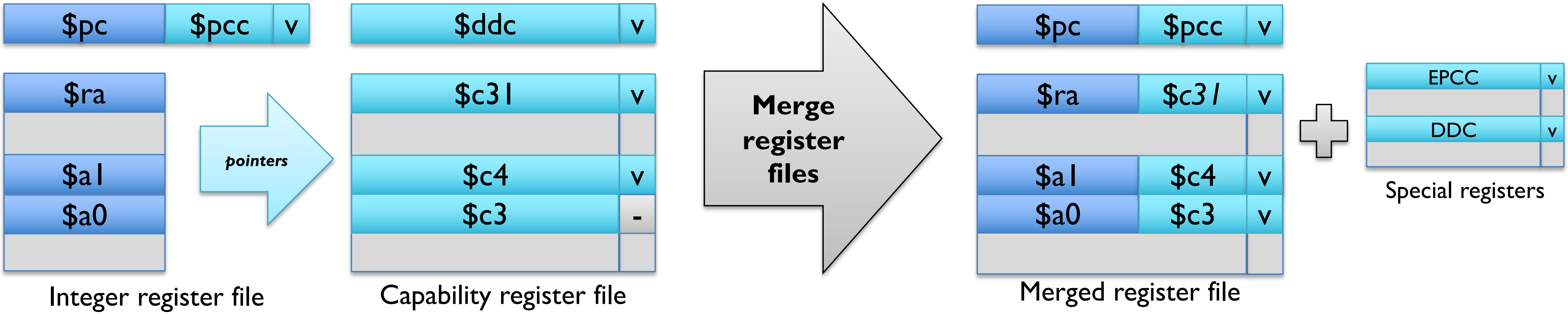
Conclusion

- New architectural primitives require rich HW and SW evaluation:
  - Primitives support many potential usage patterns, use cases
  - Applicable uses depend on compatibility, performance, effectiveness
  - Best validation approach: full hardware-software prototype
  - Co-design methodology: hardware ↔ architecture ↔ software

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

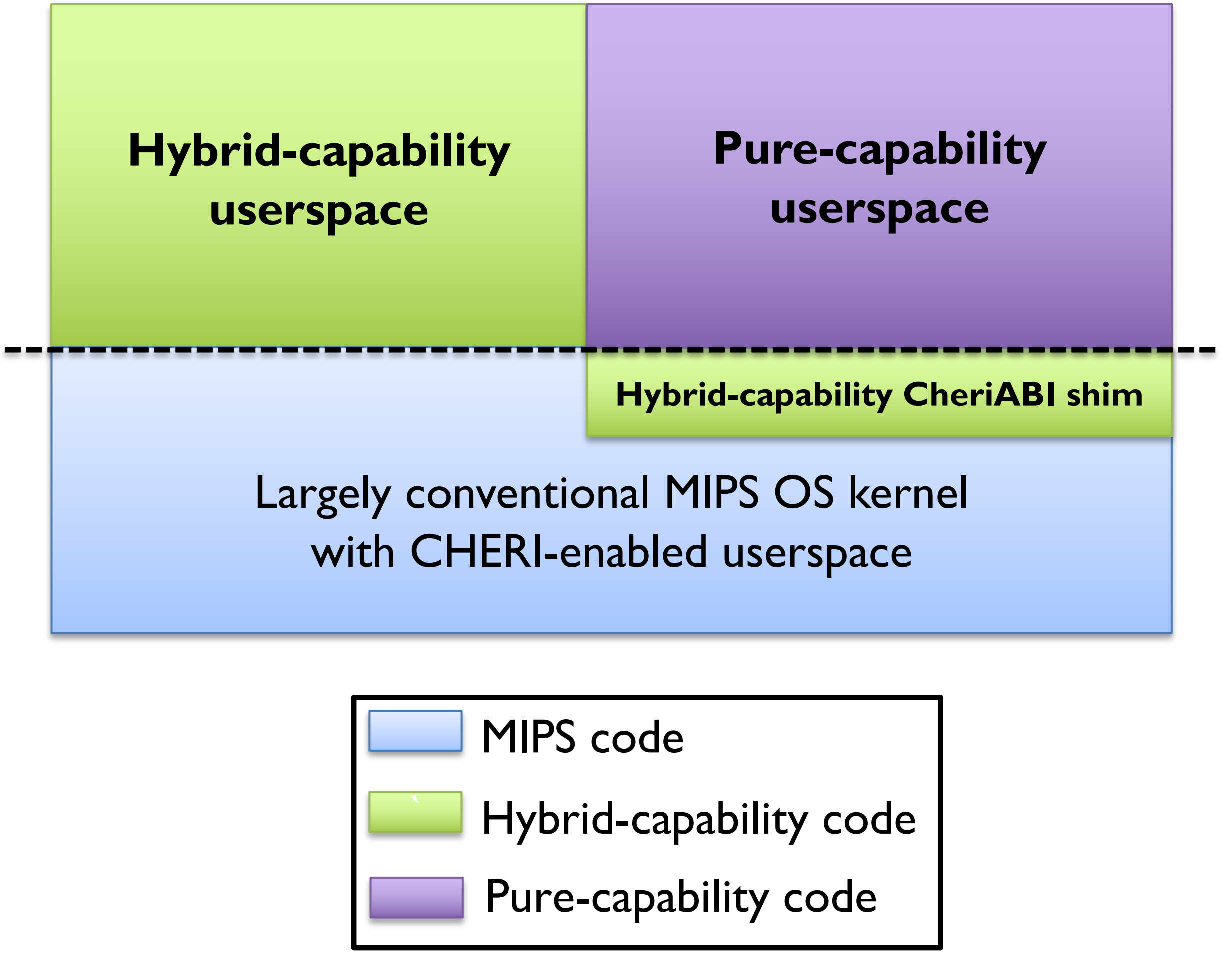
- Watson, et al. **An Introduction to CHERI**, Technical Report UCAM-CL-TR-941, Computer Laboratory, September 2019.
- Watson, et al. **Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8)**, UCAM-CL-TR-951, October 2020.
- Watson, et al. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020.

Lessons learned: Split vs. merged register files



- CHERI-MIPS has **split register files** following coprocessor conventions
- … but new register files add control logic, increasing area overhead
- Instead merge register files along the lines of 32-bit → 64-bit extension
- Key design choice in CHERI-RISC-V: Implement both approaches, evaluate

From hybrid-capability code to pure-capability code



- **n64 MIPS ABI:** hybrid-capability code
  - **Early investigation** – manual annotation and C semantics
  - Many pointers are integers (including syscall arguments, most implied VAs)
- **CheriABI:** pure-capability code
  - **More recently** – fully automatic use of capabilities wherever possible
  - All pointers, implied virtual addresses are capabilities (inc. syscall arguments)
- Now investigating pure-capability kernel

OS changes required for CheriABI

(A grand tour of low-level OS behavior)

Hybrid ABI = MIPS ABI + …

- Kernel support for tagged memory, capability context switching, etc.
- Tag-preserving libc: memory copy, memory move, sort, …
- Bounds-aware malloc(), realloc(), free(), …
- setjmp(), longjmp(), sigcontext / signal delivery, pthreads updates for capabilities
- Run-time linkage for capability-based references to globals, code, vtables, etc. (bounds, permissions, …)
- Debugging APIs such as ptrace()

CheriABI = Hybrid ABI + …

- Kernel support for pure-capability userspace
- C start-up/runtime (CSU/CRT) changes
- Initial process state: reduced initial capability registers, ELF aux args, sigcode, etc.
- Pointer arguments/return values for syscalls are now capabilities, …
- Review and fix tag preservation, integer/pointer provenance and casts
- Run-time linkage for globals, code, vtables, etc. (bounds, permissions, …)

Evaluating memory-protection compatibility

Approach: Prototype (1) "pure-capability" **CHERI C/C++ compiler** (Clang/LLVM) and (2) **full OS** (FreeBSD) that use capabilities for all explicit or implied userspace pointers

Goal: **Little or no software modification** (BSD base system + utilities)

Small changes to source files for 34 of 824 programs, 28 of 130 libraries.

Overall: modified ~200 of ~20,000 user-space C files/header

	Pointer + integer integrity, prov.	Pointer size & alignment	Monotonicity	Calling conventions	Unsupported features
BSD headers	11	6	0	2	0
BSD libraries	83	36	4	41	22
BSD programs	24	9	1	11	2

Goal: **Software that works** (BSD base + utilities test suites)

	Pass	Fail*	Skip	Total
MIPS	3501 (91%)	90	244	3835
Pure capability	3301 (90%)	122	246	3669

* Test failure investigation remains a work-in progress; we believe these can be resolved

Evaluating memory-protection impact

- Adversarial / historical vulnerability analysis
  - ✓ Pointer integrity, provenance validity prevent ROP, JOP
  - ✓ Buffer overflows: Heartbleed (2014), Cloudbleed (2017)
  - ✓ Pointer provenance: Stack Clash (2017)
- Existing test suites – e.g., BOdiagsuite (buffer overflows)

	OK	min	med	large
mips64	0	4	8	175
CheriABI	0	279	289	291
LLVM Address Sanitizer (asan) on x86	0	276	286	286

- Davis, et al. **CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment**, ASPLOS 2019.
- Key evaluation concern: reasoning about a **CHERI-aware adversary**