the second wave of

UKDF 2024 @ TNMOC

Towards Commercial Capability Machines

Simon W. Moore

UNIVERSITY OF CAMBRIDGE

With many others involved including: Robert N. M. Watson, 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

CHERI

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.

Concept from 1966 – Object Capabilities

Programming Semantics for Multiprogrammed Computations

Jack B. Dennis and Earl C. Van Horn

Massachusetts Institute of Technology, Cambridge, Massachusetts

The semantics are defined for a number of meta-instructions which perform operations essential to the writing of programs in multiprogrammed computer systems. These meta-instructions relate to parallel processing, protection of separate computations, program debugging, and the sharing among users of memory segments and other computing objects, the names of which are hierarchically structured. The language sophistication contemplated is midway between an assembly language and an advanced algebraic language.

Presented at an ACM Programming Languages and Pragmatics Conference, San Dimas, California, August 1965.

Work reported herein was supported by Project MAC, an MIT research program sponsored by the Advanced Research Projects Agency, Department of Defense, under Office of Naval Research Contract Number Nonr-4102(01). Reproduction in whole or in part is permitted for any purpose of the United States Government.

Introduction

An increasing percentage of computation activity will be carried out by multiprogrammed computer systems. Such systems are characterized by the application of computation resources (processing capacity, main memory, file storage, peripheral equipment) to many separate but concurrently operating computations.

We ca
gramme
applicat
record s
to a cent
NASA provide real time control and monitoring of manned space flights [2]. The Project MAC time-sharing system permits research workers closer interaction with the powers of automatic computation [3]. Although these are all on-line systems, multiprogramming techniques have also been

capability = unforgeable token of authority

CHERI

Capability Machines – The Early Years

Year	System	Developer	Attributes
1967	Magic Number Machine	Univ. Chicago	First capability hardware
1969	System 250	Plessy Corp.	First commercial capability computer for real-time control
1970	CAP Computer	University of Cambridge	Capability computer with microcode support
1978	System/38	IBM	First major commercial capability system; tagged capabilities
1981	iAPX432	Intel	Intel's first 32b processor! Highly-integrated (3-chip) object-based microprocessor system

From *Capability-Based Computer Systems*, H.M. Levy, Digital Press, 1984

CHERI

1970s Cambridge CAP Computer

CHERI

Cambridge CAP

- Memory capability = (permissions, base, limit)
- A program is allocated a set of capabilities
- Hardware caches 64 capabilities
  - Memory accesses check the set of capabilities for rights
  - Fully associative look-up like a Translation Look-aside Buffer (TLB)
  - Capability cache miss: microcode searches the set of capabilities
- Virtual memory and use of TLBs won
  - Simpler memory allocation including reallocation of freed pages
  - But much less precise protection

CHERI – A Modern Capability Architecture

CHERI = Capability Hardware Enhanced RISC Instructions

Motivation: Memory Safety

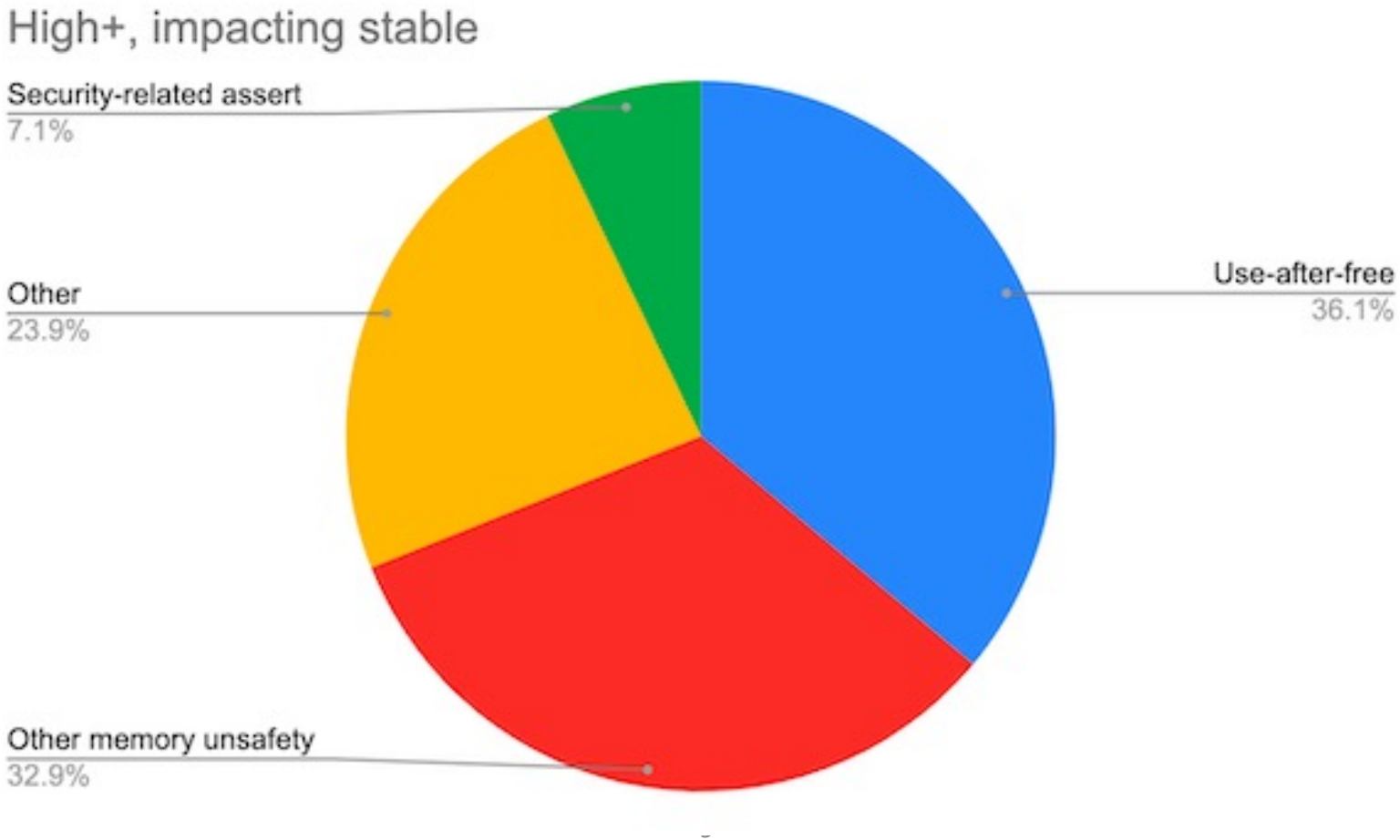
- *"Buffer overflows have not objectively gone down in the last 40 years. The impact of buffer overflows have if anything gone up."*
  Ian Levy, Technical Director at NCSC, 2022

- Matt Miller from MS Response Center @ BlueHat 2019:
  - From 2006 to 2018, year after year, 70% MSFT CVEs are memory safety bugs

Example – Chromium Browser Safety

"70% of our serious security bugs are memory safety problems"

www.chromium.org/Home/chromium-security/memory-safety



CHERI

Software Memory Safety

Executive summary

Modern society relies heavily on software-based automation, implicitly trusting developers to write software that operates in the expected way and cannot be compromised for malicious purposes. While developers often perform rigorous testing to prepare the logic in software for surprising conditions, exploitable software vulnerabilities are still frequently based on memory issues. Examples include overflowing a memory buffer and leveraging issues with how software allocates and de-allocates memory. Microsoft® revealed at a conference in 2019 that from 2006 to 2018 70 percent of their vulnerabilities were due to memory safety issues. [1] Google® also

CISA, NSA, FBI, and US ally cybersecurity agencies recommend CHERI

April 2023

Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Security-by-Design and -Default

Publication: April 13, 2023

Cybersecurity and Infrastructure Security Agency

NSA | FBI | ACSC | NCSC-UK | CCCS | BSI | NCSC-NL | CERT NZ | NCSC-NZ

Disclaimer: *This document is marked TLP:CLEAR. Disclosure is not limited. Sources may use TLP:CLEAR when information carries minimal or no foreseeable risk of misuse, in accordance with applicable rules and procedures for public release. Subject to standard copyright rules, TLP:CLEAR information may be distributed without restriction. For more information on the Traffic Light Protocol, see http://www.cisa.gov/tlp/.*

products. Threat models consider a product's specific use-case and enables development

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

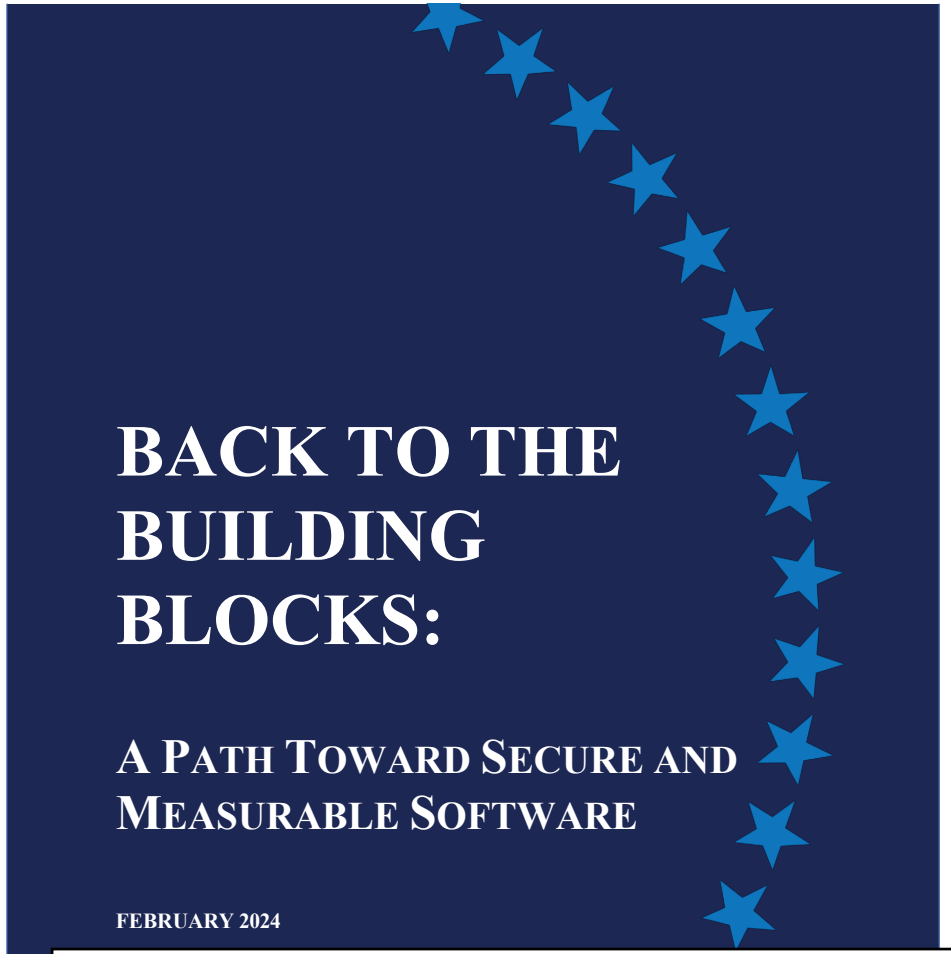
- **Memory safe programming languages (SSDF PW.6.1):** Prioritize the use of memory safe languages wherever possible. The authoring agencies acknowledge that other memory specific mitigations, such as address space layout randomization (ASLR), control-flow integrity (CFI), and fuzzing are helpful for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

- **Static and dynamic application security testing (SAST/DAST) (SSDF PW.7.2, PW.8.2):**

CHERI

The White House Technical Report

February 2024



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

CHERI

How can we do better?

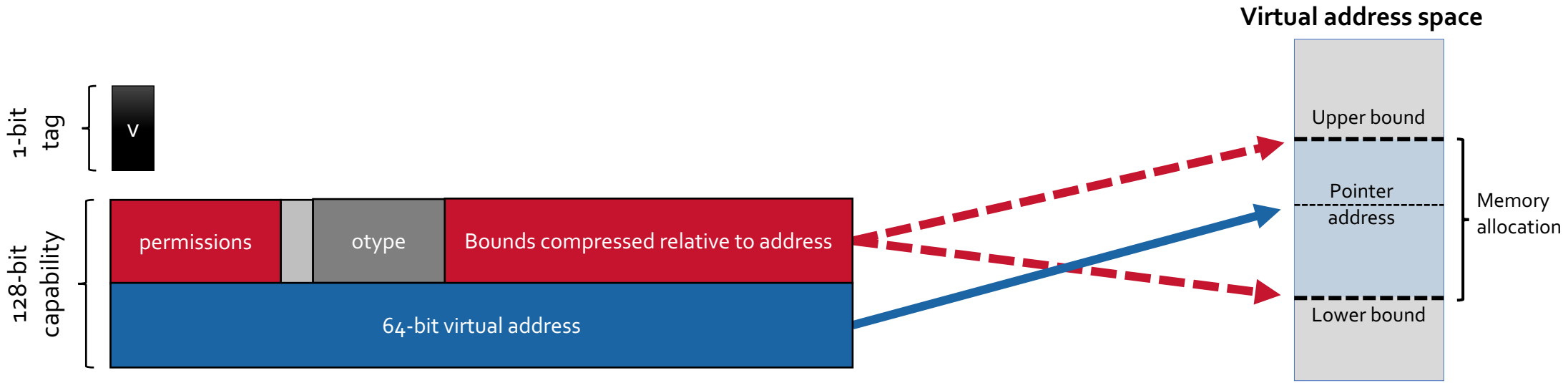
1. Use memory safe languages like Rust?
   - Great for new code
   - Impractical for the vast body of legacy code?

2. Our approach: make C/C++ and other languages memory safe using CHERI capabilities
   - Recompile your code for a CHERI enhanced processor to get memory safety
   - Can also make unsafe Rust much safer!

We've now ported more lines of C/C++ code to make it memory safe than exist lines of Rust code on GitHub!

CHERI

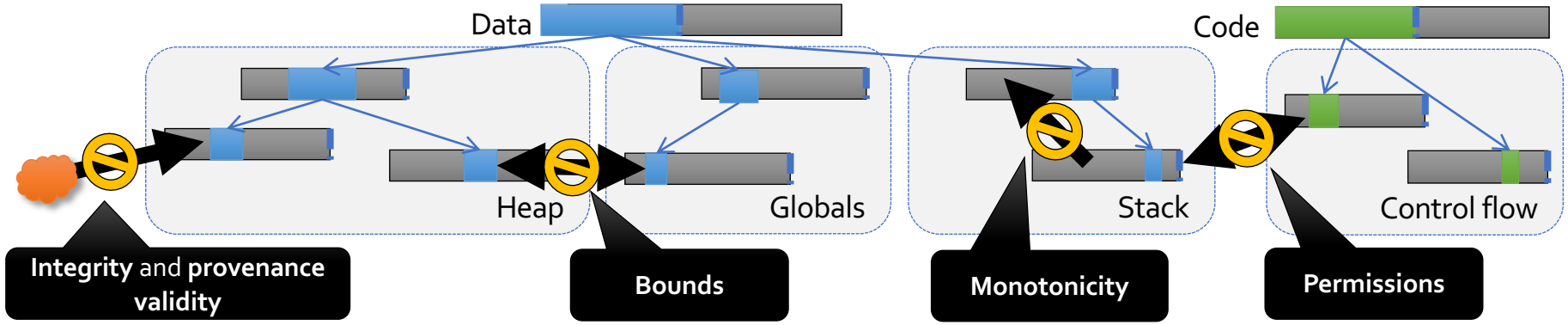
CHERI 128-bit capabilities



- **Capabilities** extend **integer memory addresses** including jump targets
- **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

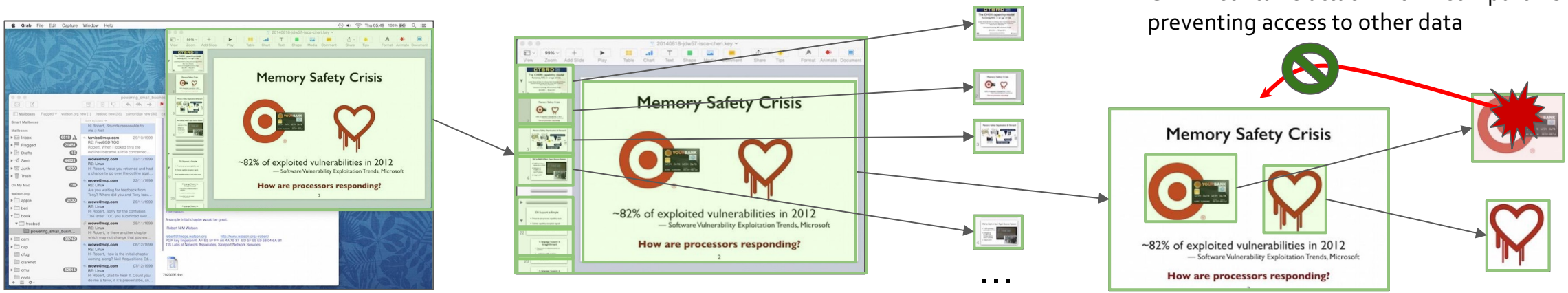
CHERI enforces protection semantics for pointers



- **Integrity** and **provenance validity** ensure that valid pointers are derived from other valid pointers via valid transformations; **invalid pointers cannot be used**
- **Bounds** prevent pointers from being manipulated to access the wrong object
- **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

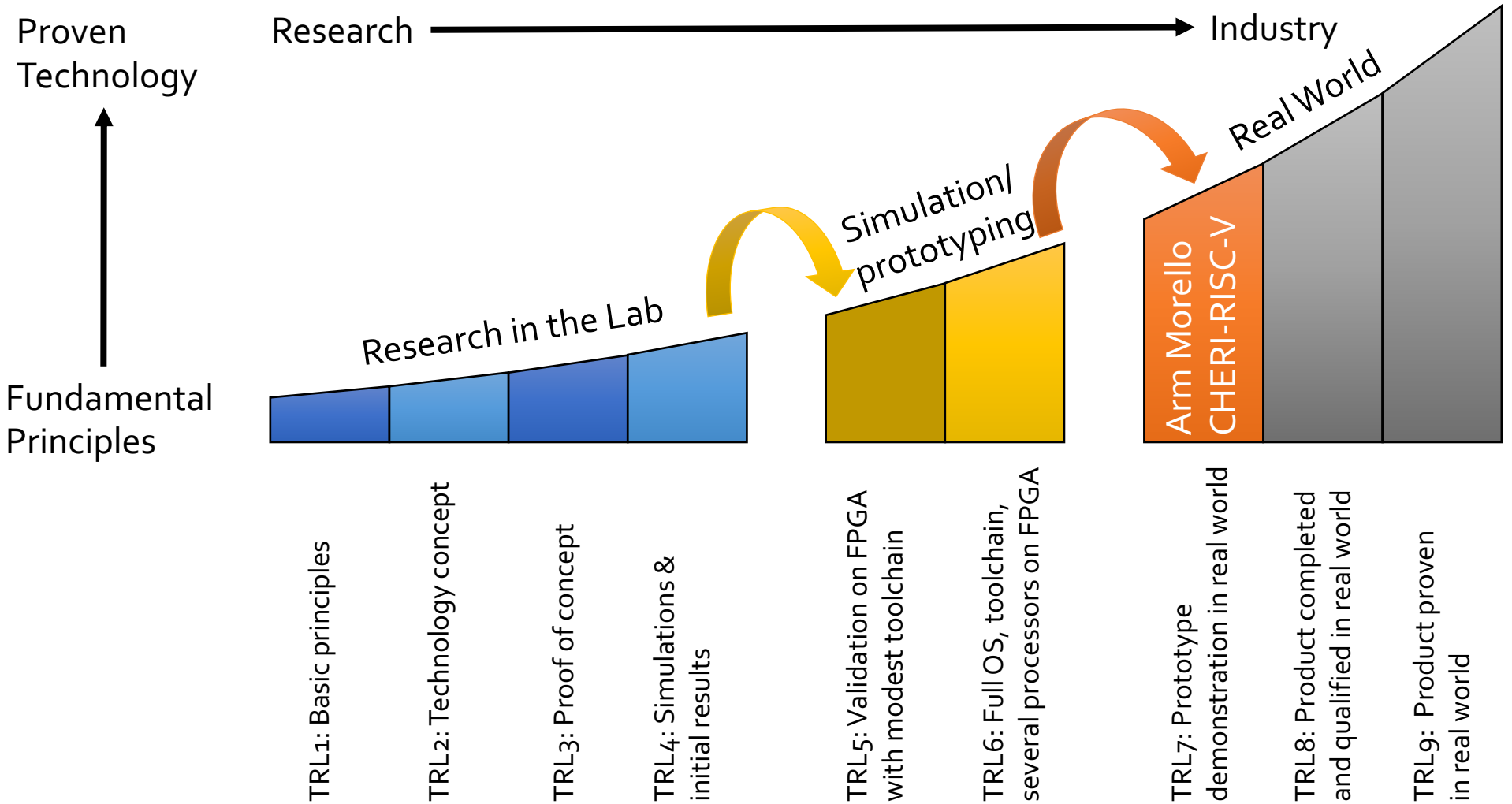
Software compartmentalization at scale



- 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 improves each of these by at least an order of magnitude

CHERI

Bridging the commercialisation chasm

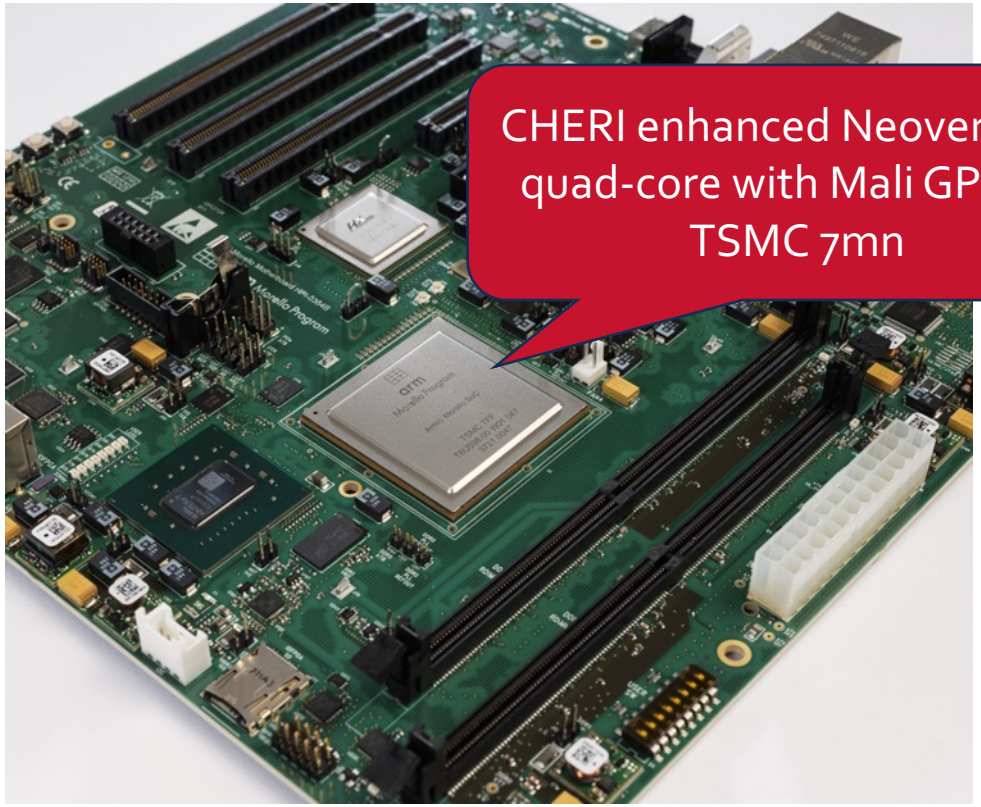


CHERI

arm Morello Demonstrator Board

Innovate UK – Digital Security by Design Initiative

arm Morello Program



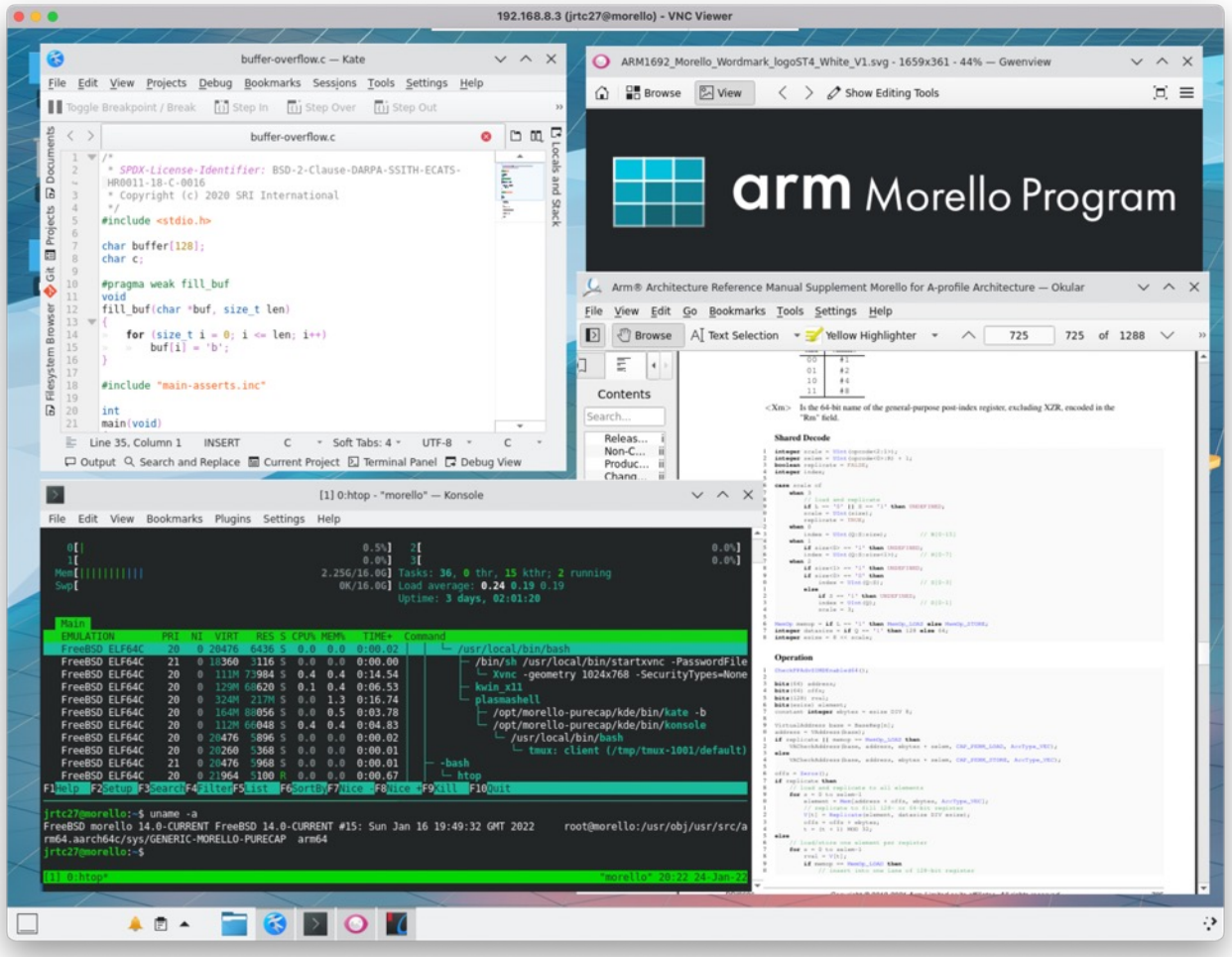
CHERI enhanced Neoverse N1 quad-core with Mali GPU on TSMC 7mn

https://www.arm.com/architecture/cpu/morello

No IP encumberment due to "Capability Essential IP" agreement between ARM and University of Cambridge

CHERI

CHERI desktop ecosystem study: Key outcomes



Developed:

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

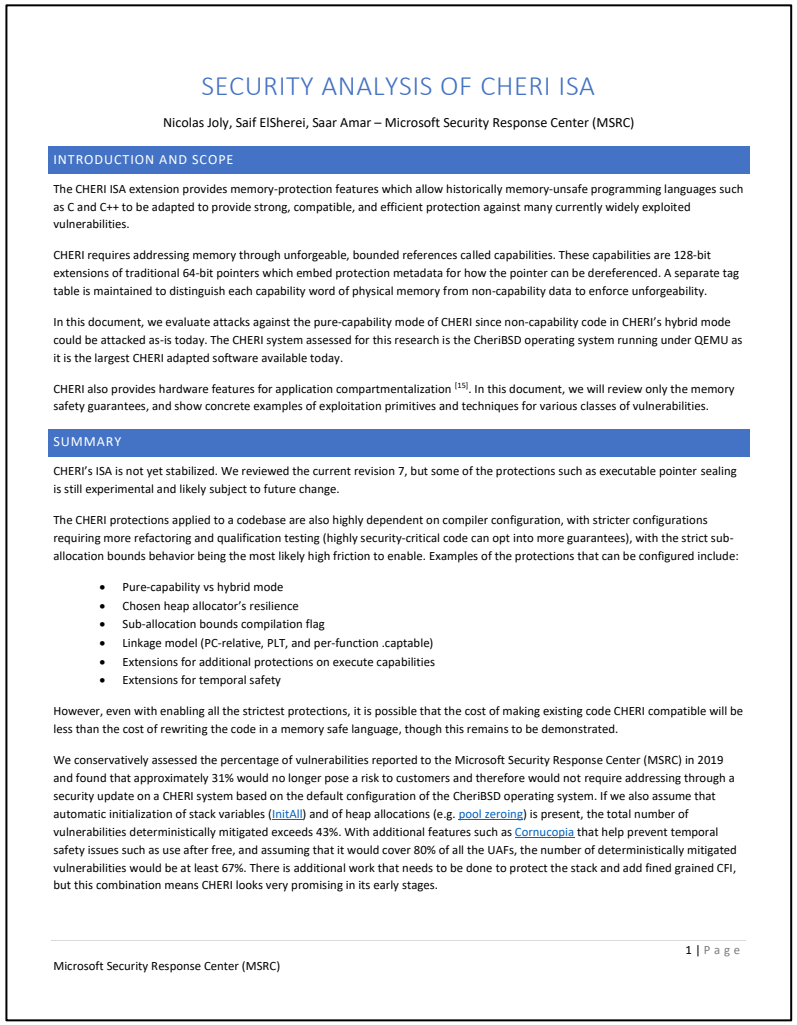
Evaluation results:

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

CHERI

http://www.capabilitieslimited.co.uk/pdfs/20210917-capltd-cheri-desktop-report-version1-FINAL.pdf

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
- CHERI, "in its current state, and combined with other mitigations, it would have **deterministically mitigated at least two thirds of all those issues**"

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

CHERI

Microsoft announce CHERIoT

https://www.microsoft.com/en-us/research/publication/cheriot-rethinking-security-for-low-cost-embedded-systems/

CHERIoT: Rethinking security for low-cost embedded systems

Saar Amar, Tony Chen, David Chisnall, Felix Domke, Nathaniel Filardo, Kunyan Liu, Robert Norton-Wright, Yucong Tao, Robert N. M. Watson, Hongyan Xia

MSR-TR-2023-6 | February 2023
Published by Microsoft

Developer info at cheriot.org

Small embedded cores have little area to spare for security features and yet must often run code written in unsafe languages and, increasingly, are exposed to the hostile Internet. CHERIoT (Capability Hardware Extension to RISC-V for Internet of Things) builds on top of CHERI and RISC-V to provide an ISA and software model that lets software depend on object-granularity spatial memory safety, deterministic use-after-free protection, and lightweight compartmentalization exposed directly to the C/C++ language model. This can run existing embedded software components on a clean-slate RTOS that scales up to large numbers of isolated (yet securely communicating) compartments, even on systems with under 256 KiB of SRAM.

PDF

Groups

Azure Research

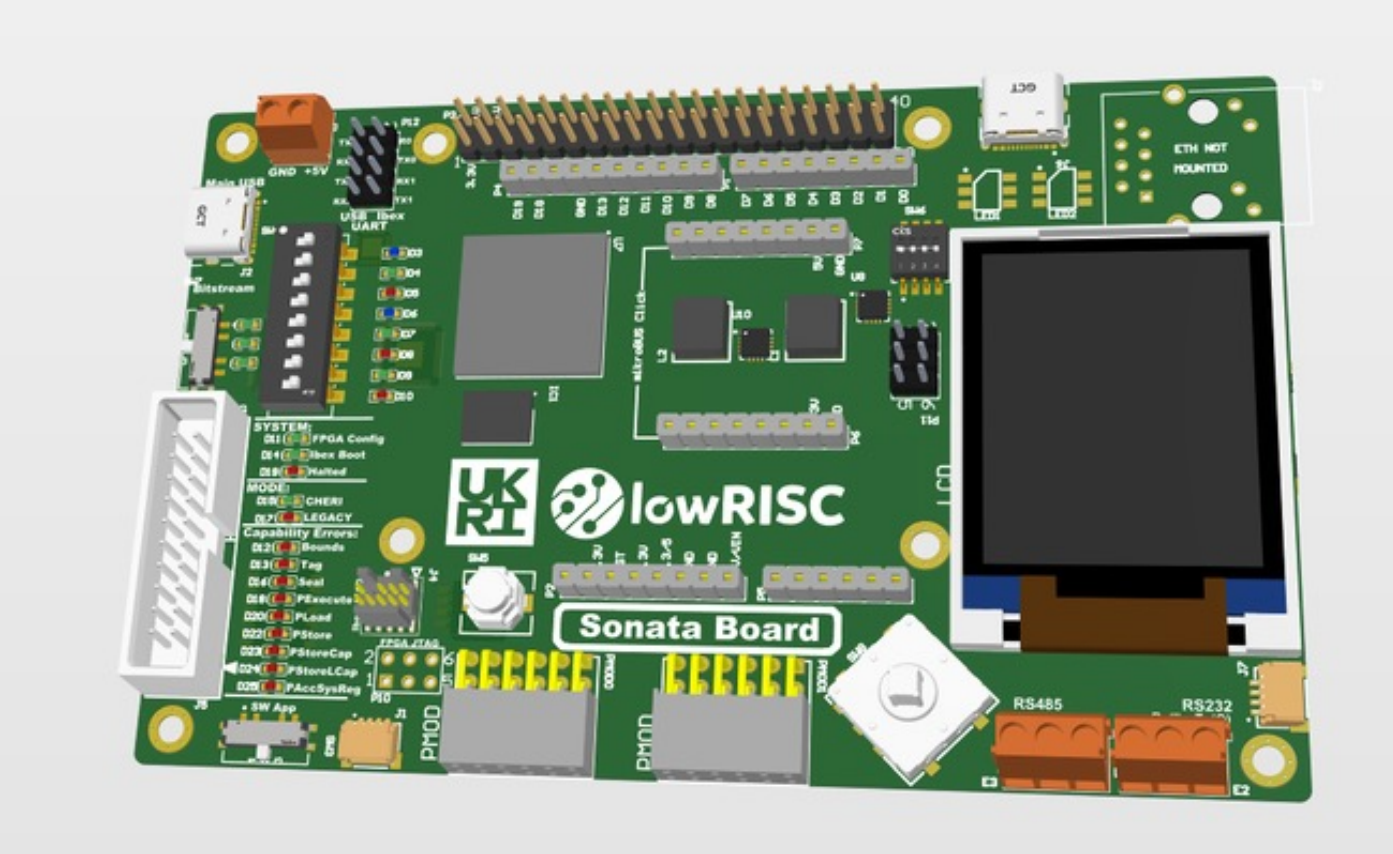
Projects

Portmeirion

CHERI

lowRISC – Sunburst

- FPGA platform for CHERIoT
- Builds on lowRISC OpenTitan silicon root of trust
- Funded by Innovate UK, Digital Security by Design Initiative



https://www.sunburst-project.org/

CHERI

Codasip announce commercial CHERI-RISC-V

https://codasip.com/solutions/riscv-processor-safety-security/cheri/

CHERI

CHERI-RISC-V Open-Source Stack

- CHERI-RISC-V developed open source:
  - Documentation (ISA ref, architecture overview, etc)
  - Specification in Sail
  - Simulators: Spike, Qemu
  - Clang/LLVM toolchain
  - OS support: CheriBSD, CheriFreeRTOS, CheriRTEMS
  - Hardware implementations
    - 3-stage, 5-stage and OoO cores on FPGA including AWS F1

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

Links to research:
http://cheri-cpu.org/
links to:
https://www.cl.cam.ac.uk/ research/security/ctsrd/

CHERI-RISC-V extension for comment:
https://github.com/riscv/riscv-cheri

CHERI

Summary

- Software level
  - Deterministic non-bypassable memory safety (spatial and temporal)
  - High-performance scalable compartmentalization
  - Easy to adopt – recompile with very few fixes beyond bug fixes
- Architecture level
  - Detailed implementations on RISC-V and ARM ISAs; outline x86 extension
- Microarchitectural level
  - Demonstrably implementable
  - No new: indirections, tables, exception paths, register renaming, etc.
  - No performance impact on legacy (non-CHERI) code
  - Main performance hit for CHERI code is memory footprint – around 5% for SPEC
  - Removes the need for many software counter measures + scalable and highly performant compartmentalization

CHERI

CHERI Tech 2024 in Cambridge

23rd April 2024

Time	Topic	Speaker
09:00 - 09:30	Arrival and coffee	
09:30 - 11:00	Session 1: Industry Applications, Chair: Robert N. M. Watson	
09:30 - 09:40	Welcome, introduction, and brief update	Robert N. M. Watson
09:40 - 10:10	CHERI-RISC-V standardisation	Tariq Kurd
10:10 - 10:35	AutoCHERI	Hardeep Chahal
10:35 - 11:00	RESAuto	Peter Davies
11:00 - 11:30	Coffee break	
11:30 - 12:30	Session 2: Compartmentalisation, Chair: Franz A. Fuchs	
11:30 - 11:50	CAP-TEE	David Oswald
11:50 - 12:10	Using library compartmentalisation in user-space	Nick Connolly
12:10 - 12:30	FlexCAP	Pierre Oliver
12:30 - 14:00	Lunch break	
14:00 - 15:40	Session 3: Bringing CHERI to the next level, Chair: Peter Sewell	
14:00 - 14:25	Vulnerability detection	Daniel King
14:25 - 14:50	CAPCelerate	Matt Naylor
14:50 - 15:15	Rust for CHERI	Sarah Harris
15:15 - 15:40	Formal Verification of CHERI processors	Tom Melham
15:40 - 16:15	Coffee break	
16:15 - 17:30	Session 4: Ecosystem, Chair: Simon W. Moore	
16:15 - 16:35	Capable VMs	Jeremy Singer
16:35 - 16:55	Combining CHERIoT and Morello	David Chisnall
16:55 - 17:15	Chromium and V8 on Morello	Graeme Jenkinson
17:15 - 17:30	Morello Linux Project	Vincenzo Frascino

CHERI