

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 Marketos, J. Edward Maste, Alfredo Mazinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch,
Edward Napierala, George Neville-Neil, Kyndylan Nienhuis, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu,
Allison Randal, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Thomas Sewell, Stacey Son,
Ian Stark, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Jonathan Woodruff, Hongyan Xia, Vadim Zaliva,
and Bjoern A. Zeeb

SRI International, Capabilities Limited, and the University of Cambridge
CHERI Conference – 12 November 2024

Approved for public release; distribution is unlimited.

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

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

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

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 prototype processor, board announced shipped 2022, supported by UKRI
 - Microsoft CHERIoT (RISC-V) Ibex core announced Sep 2022 and open sourced in February 2023; lowRISC Sonata board announced Sep 2023; Cudasip IP core products announced October 2023
- Today's talk:
 - What is CHERI and how does it change software security?
 - Transition efforts including Arm, Google, Microsoft, and beyond ...
- <http://www.cheri-cpu.org/>
- Watson, et al., **CHERI: Hardware-Enabled C/C++ Memory Protection at Scale**, IEEE Security and Privacy Magazine, July-August 2024.



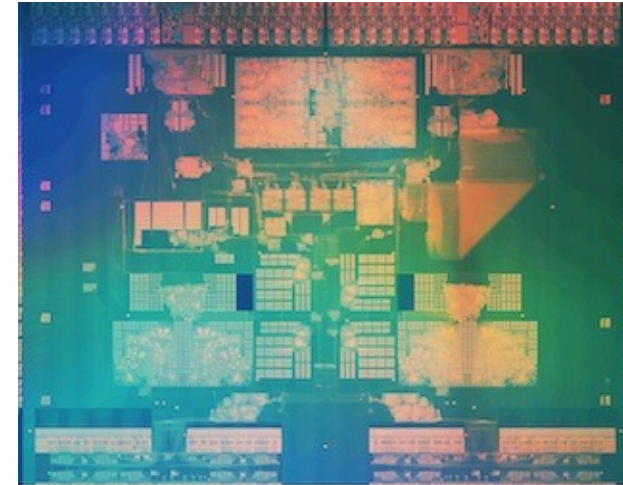
An early experimental FPGA-based CHERI tablet prototype running the CheriBSD operating system and applications, Cambridge, 2013.



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

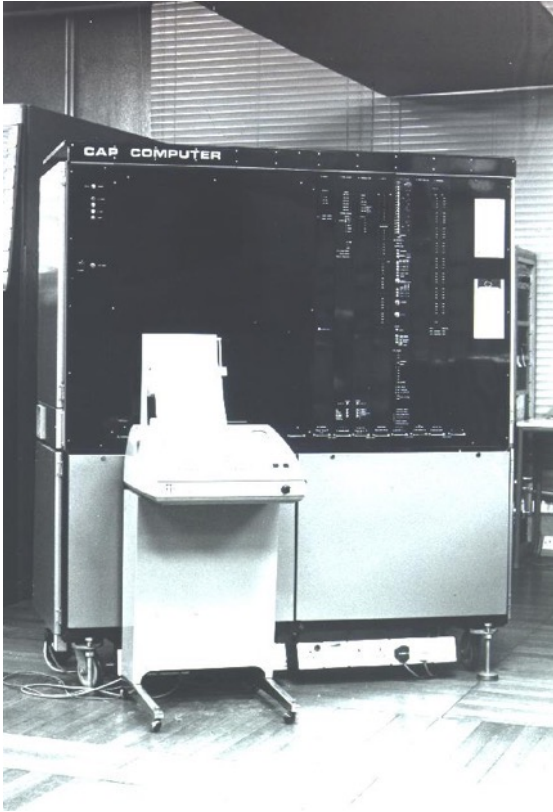
What is CHERI?

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



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

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

CISA, NSA, FBI, NCSC, and ally cybersecurity organisations 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 teams to fortify products. Finally, senior leadership should hold teams accountable for

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~~

8 • **Static and dynamic application security testing (SAST/DAST) (SSDF PW.7.2, PW.8.2):**
CISA | NSA | FBI | ACSC | NCSC-UK | CCCS | BSI | NCSC-NL | CERT NZ | NCSC-NZ
TLP:CLEAR

BACK TO THE BUILDING BLOCKS:

A PATH TOWARD SECURE AND MEASURABLE SOFTWARE

FEBRUARY 2024



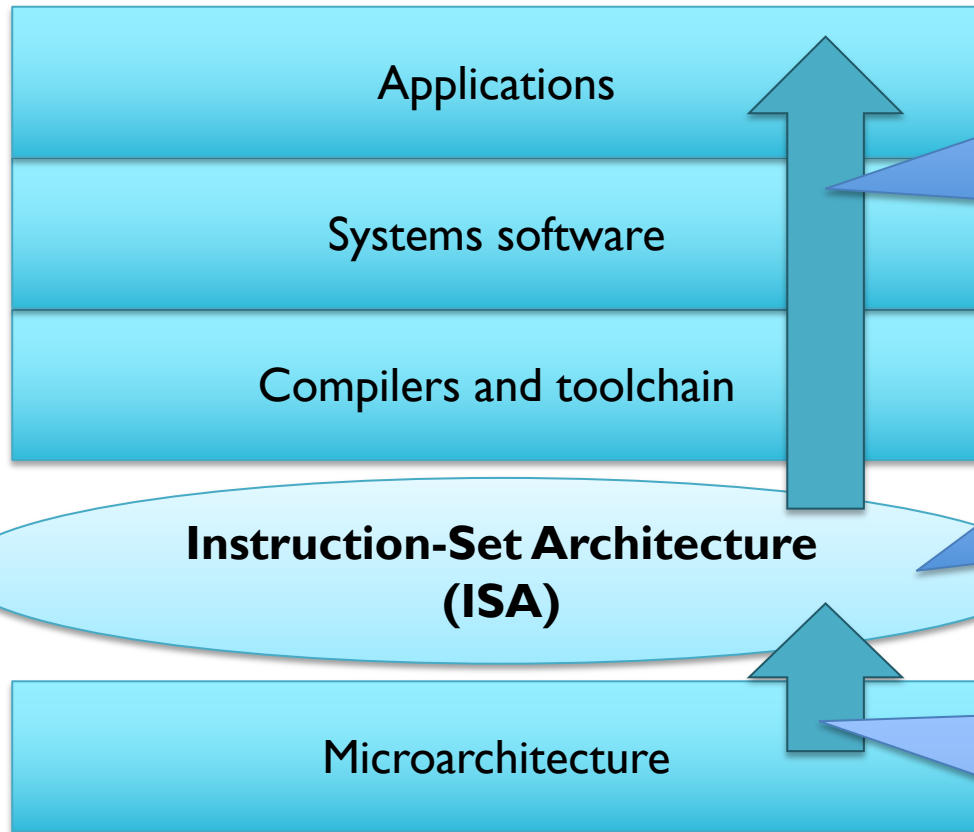
THE WHITE HOUSE
WASHINGTON

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 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**

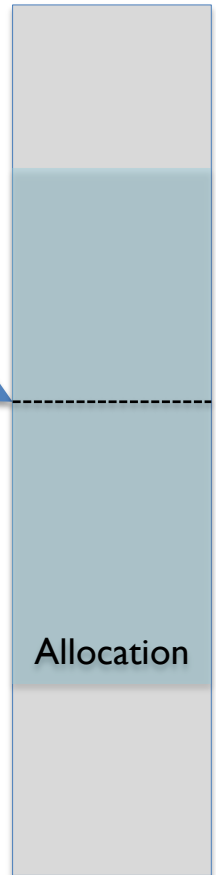
CHERI capabilities are an **architectural primitive** that compilers, systems software, and applications use to constrain their own future execution

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**

The weakness: Pointers

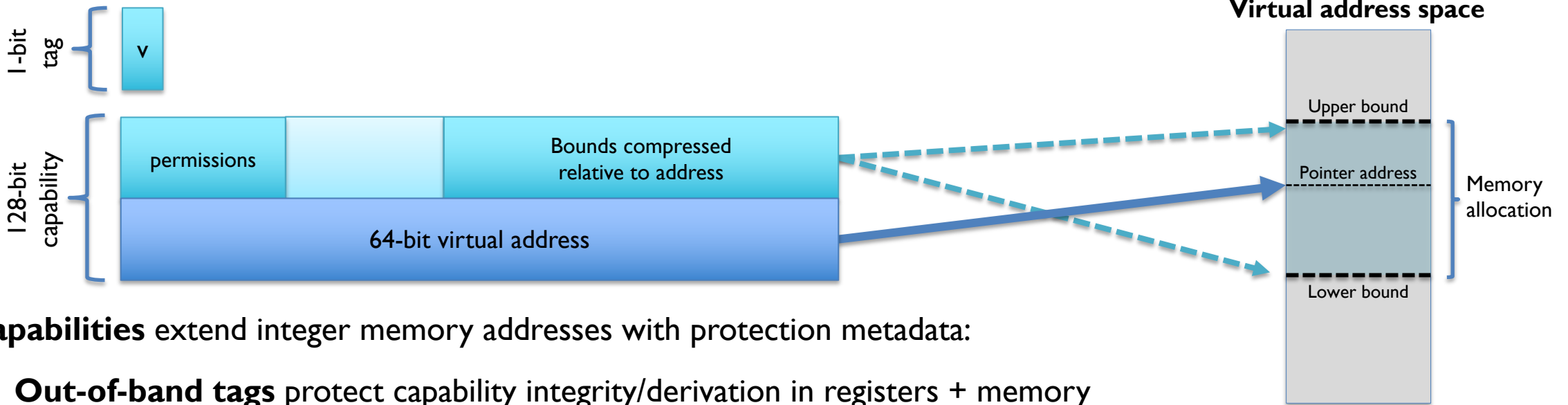


- Implemented as **integer virtual addresses (VAs)**
- (Usually) point into **allocations, mappings**
 - **Derived** from other pointers via integer arithmetic
 - **Dereferenced** via jump, load, store
- **No integrity protection** – can be injected/corrupted
- **Arithmetic errors** – out-of-bounds leaks/overwrites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



Virtual
address
space

CHERI | 28-bit capabilities (64-bit, MMU-enabled)



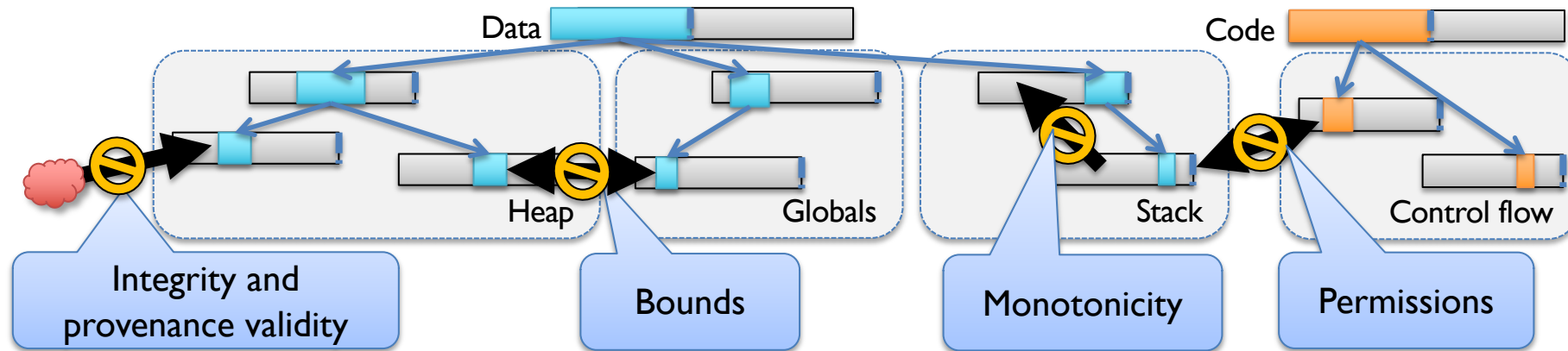
Capabilities extend integer memory addresses with protection metadata:

- **Out-of-band tags** protect capability integrity/derivation in registers + memory
 - Dereferencing an invalid capability (tag value of zero) throws an exception
 - Overwriting a capability in memory clears its validity tag
- **Bounds** and **permissions** authorize access to memory
 - Dereferencing a capability outside of its bounds, permissions, etc., throws an exception
- **Guarded manipulation** controls how capability values themselves may be manipulated

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

E.g., enforcing **provenance validity** and **monotonicity**

Capability semantics applied to C/C++ pointers

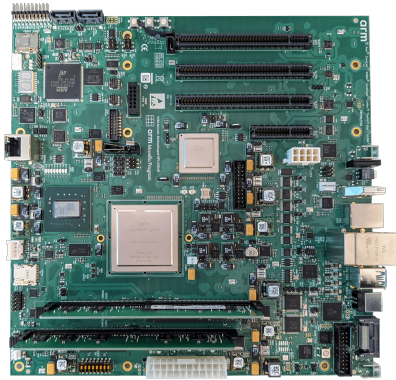


- **Tags** protect the **integrity** and **provenance validity** of pointers by:
 - Constraining manipulation, detecting corruption, and preventing injection (e.g., via the network)
 - Enabling accurate and deterministic detection, efficient tracking and revocation (i.e., temporal safety)
 - **Bounds**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

CHERI MICROARCHITECTURE AND PROTOTYPES

CHERI demonstrated at a range of scales

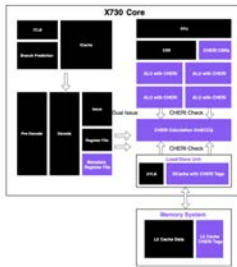
Mobile devices, data centers



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



Automotive, embedded, high-end IoT



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



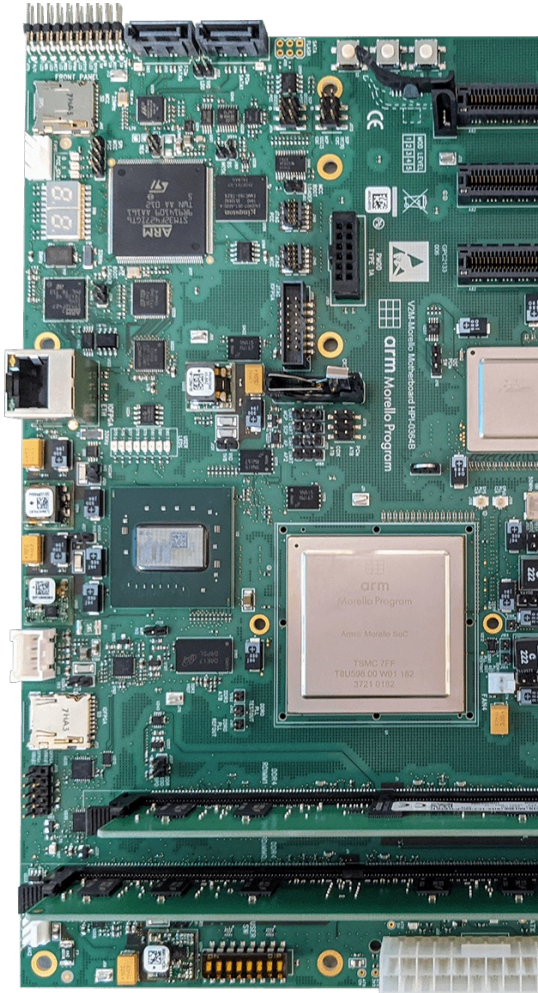
IoT, roots of trust



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



Arm Morello (2022)



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.

For the purpose of open access, the authors have applied a Creative Commons Attribution (CC BY) license.

IEEE Micro Journal 2023

- £119M government, academia, and industrial research program led by UK Research and Innovation (UKRI)
 - Announced partners: Arm, Google, Microsoft
 - 15+ UK universities with research grants
 - 70+ funded business incubation projects
- Baseline for design: Neoverse N1 core
 - 2.5GHz quad-core, superscalar
 - Implements CHERI extensions
 - Runs full CHERI-enabled software stacks
 - Definitely a prototype, but a very powerful one!
- Roughly a thousand chips manufactured for use by research + development labs

Microsoft CHERIoT core (2023)

CHERIoT: Complete Memory Safety for Embedded Devices

Saar Amar*
saarama5@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

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



This work is licensed under a Creative Commons Attribution International 4.0 License.

MICRO '23, October 28–November 01, 2023, Toronto, ON, Canada
© 2023 Copyright held by the owner/author(s).
ACM ISBN 978-8-4007-6329-4/23/10.
<https://doi.org/10.1145/3613424.3614266>

practicality of an end-to-end IoT application. The implementation shows that full memory safety for compartmentalized embedded systems is achievable without violating resource constraints or real-time guarantees, and that hardware assists need not be expensive, intrusive, or power-hungry.

ACM Reference Format

Saar Amar, David Chisnall, Tony Chen, Nathaniel Wesley Filardo, Ben Laurie, Kunyan Liu, Robert Norton, Simon W. Moore, Yucong Tao, Robert N. M. Watson, and Hongyan Xia. 2023. CHERIoT: Complete Memory Safety for Embedded Devices. In *66th 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

- CHERI-extended Ixex microcontroller
 - Microcontroller used in OpenTitan, etc.
 - CHERI-RISC-V tuned for microcontrollers
 - Clean-slate memory-safe, compartmentalized embedded OS for high-exposure applications
 - CHERI extensions for revocation in SRAM
 - Open sourced in February 2023
- Collaboration across Microsoft Research, MSRC, Azure Silicon, and Azure Edge + Platform
- Various in-progress productizations for embedded use cases — whether IoT, roots of trust, etc.

Announced, in-progress, CHERI-RISC-V adoption



First production CHERI-RISC-V
silicon to ship in 2025

- **Microsoft** open-source CHERIoT-Ibex 32-bit microcontroller IP core
- **Google** CHERI-enabled open-source ML accelerators
- **lowRISC** CHERIoT-based Sonata FPGA development board
- **SCI Semiconductor** CHERIoT-based embedded SoCs
- **Codasip** proprietary 64-bit, MMU-enabled application core IP
- Active **RISC-V standardization** effort in RISC-V International

HOW SOFTWARE WORKS ON CHERI

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 $\frac{2}{3}$ of memory-safety vulnerabilities mitigated
- Generally modest overhead (0%-5%, some pointer-dense workloads higher)

2. Scalable software compartmentalization

- Multiple software operational models from objects to processes
- Increases exploit chain length: Attackers must find and exploit more vulnerabilities
- Orders-of-magnitude performance improvement over MMU-based techniques (<90% reduction in IPC overhead in early FPGA-based benchmarks)

CHERI C/C++ MEMORY PROTECTION

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

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

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

Memory-safe CHERI C/C++

Technical Report

UCAM-CL-TR-947
ISSN 1476-2986

Number 947



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, 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 comparable in [scope, cost] to the 32-bit to 64-bit transition
- We have adapted in excess of 150MLoC of open-source C/C++ code to strong memory safety with minimal or no changes to most code
- Watson, et al. **CHERI C/C++ Programming Guide**, UCAM-CL-TR-947, June 2020

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 .cappable)
- 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 fine grained CFI, but this combination means CHERI looks very promising in its early stages.

1 | Page

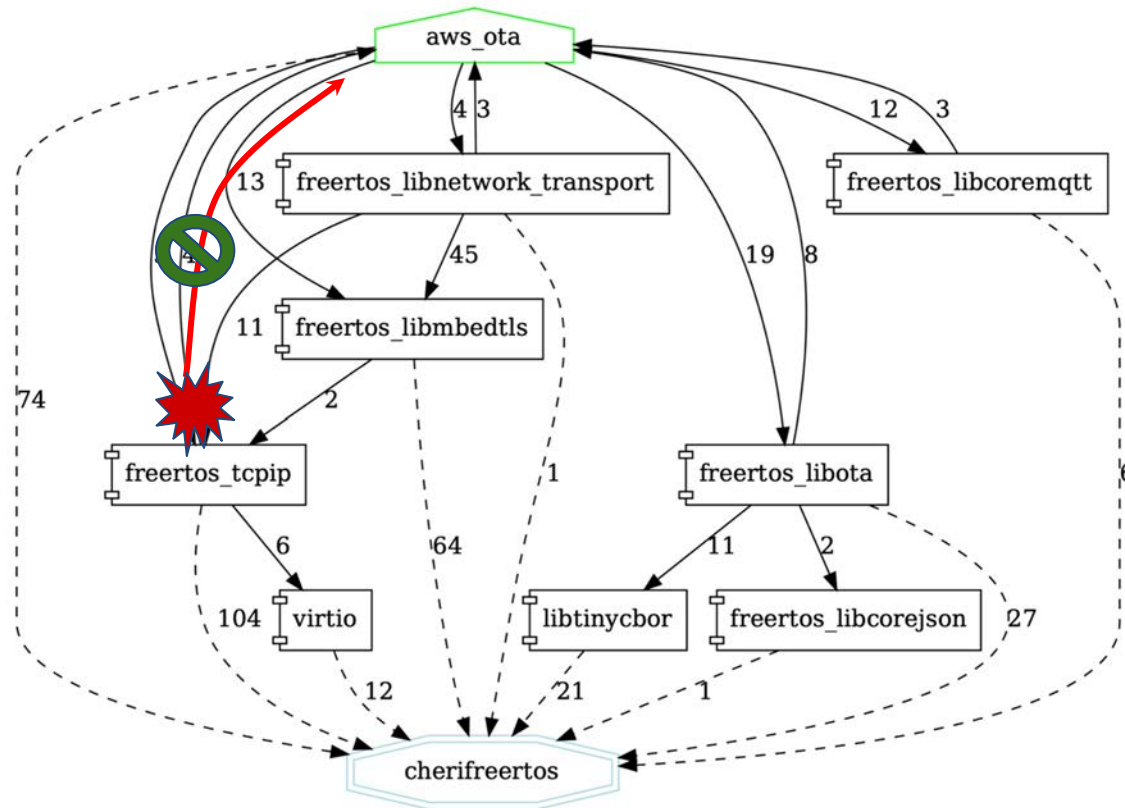
Microsoft Security Response Center (MSRC)

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

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

CHERI SOFTWARE COMPARTMENTALISATION

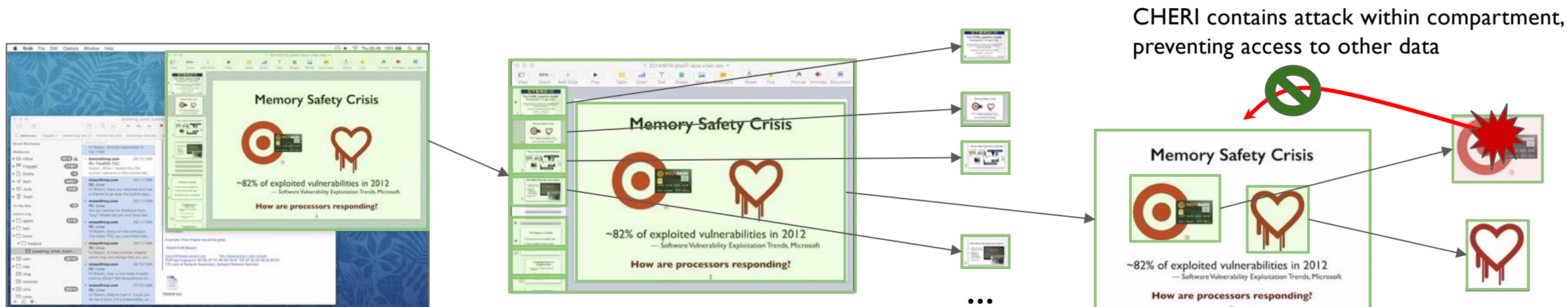
What is software compartmentalization?



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

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

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 one order of magnitude, and often two

A COMPLETE CHERI-ENABLED SOFTWARE STACK

2021 desktop pilot study results

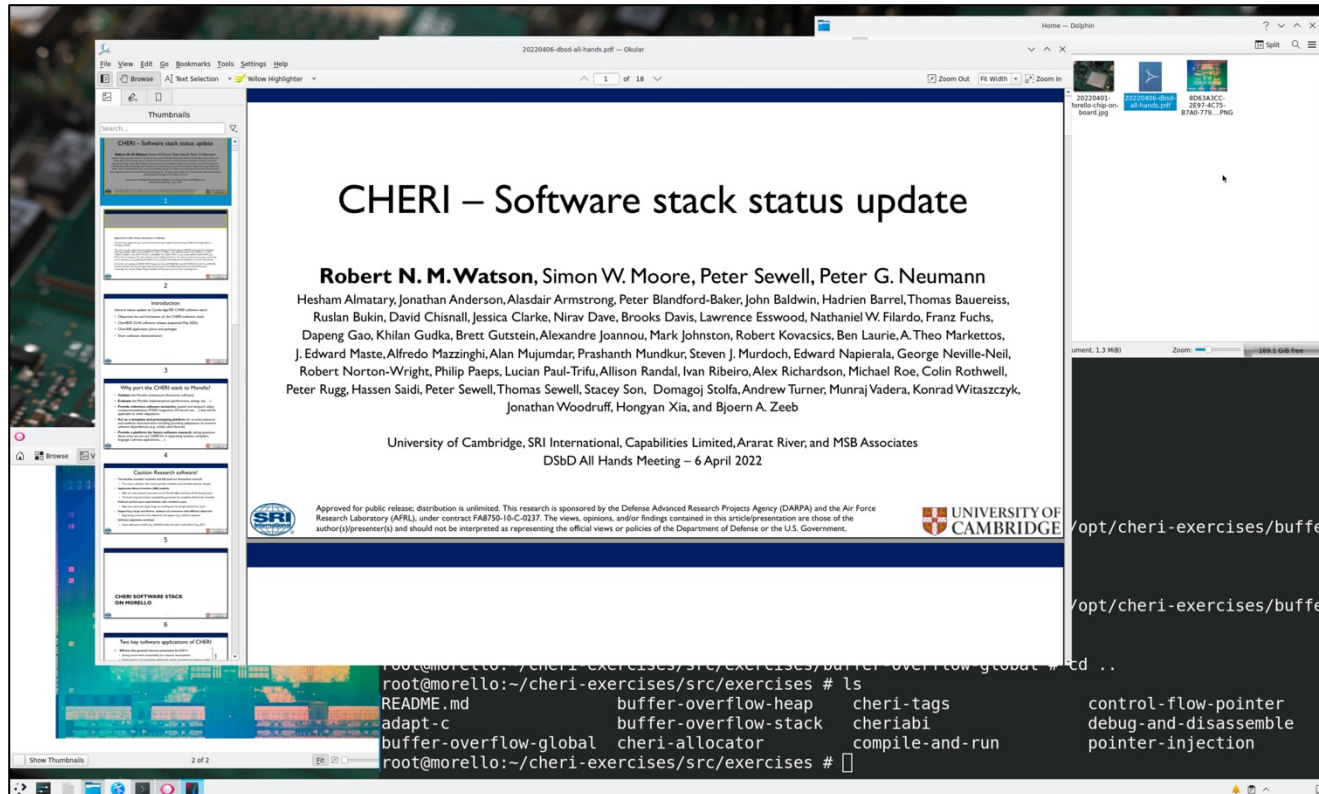
Developed:

- **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

Had to reverse engineer “de facto” threat models for open-source software as not well documented; key definition “required patches”

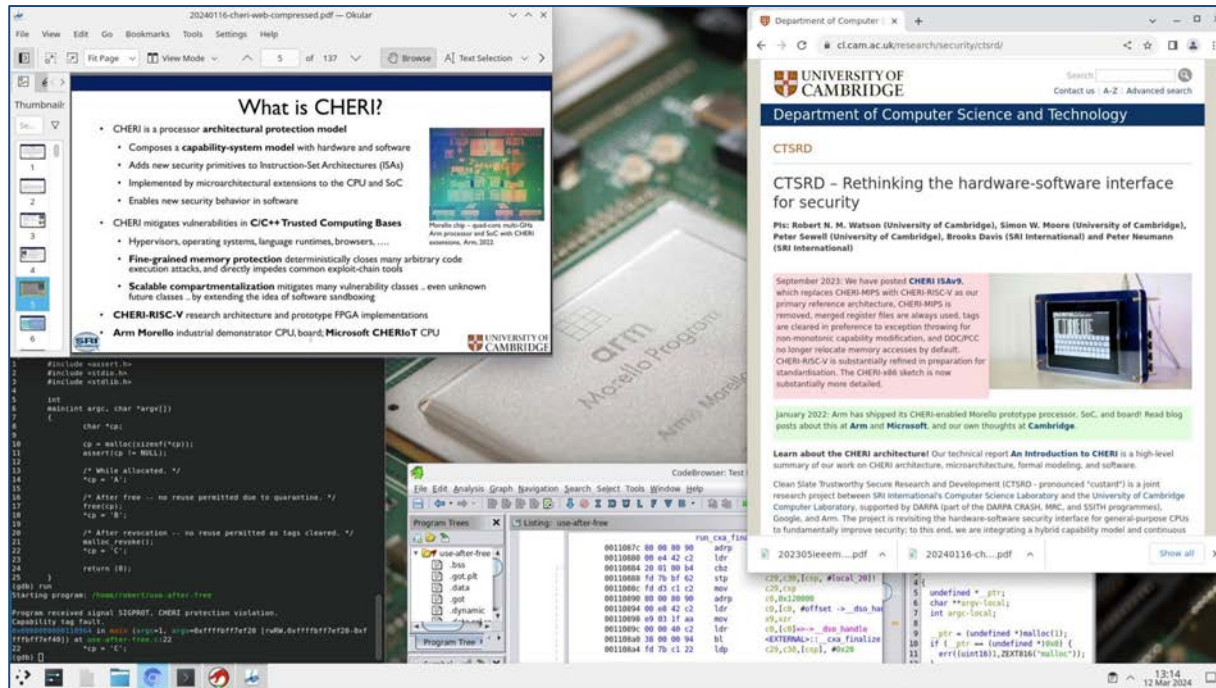


2024.05 Morello memory-safe desktop stack

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

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

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



CHERI and legacy applications side-by-side on Morello

Enables incremental application migration to memory safety

Memory-safe
PDF viewer

Memory-safe
desktop
environment

Memory-safe
terminal window
and commands

Memory-safe
OS kernel

Legacy 64-bit
Arm
Chromium
browser

Legacy 64-bit
Arm JVM

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

```
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char *argv[])
{
    char *cp;
    cp = malloc(sizeof(*cp));
    assert(cp != NULL);
    /* While allocated, */
    *cp = 'A';
    /* After free -- no reuse permitted due to quarantine. */
    cp = 'B';
    /* After revocation -- no reuse permitted as tags cleared. */
    malloc_revoked();
    *cp = 'C';
    return (0);
}

(gdb) run
Starting program: /home/robert/ssa/after-free
Program received signal SIGPROT, CHERI protection violation.
Capability tag fault:
0000000000000000 in main (argc=1, argv=0xfffffff7ef20 [rW, 0xfffffff7ef20, 0x7ffff7ef40]) at ssa/after-free.c:22
22      *cp = 'C';
(gdb) 
```

CodeBrowser: Test

Listing: use-after-free

```
run_cxa_finalize
0011087c 00 00 80 90  adrp
00110880 00 e4 42 c2  ldr
00110884 20 01 00 b4  cbz
00110888 fd 7b bf 62  stp
0011088c fd d3 c1 c2  mov
00110890 00 00 80 90  adrp
00110894 00 e0 42 c2  ldr
00110898 e9 03 1f aa  mov
0011089c 00 00 40 c2  ldr
001108a0 38 00 00 94  bl
001108a4 fd 7b c1 22  ldp
```

Memory-safe OS kernel and libraries

DEMONSTRATION

CONCLUSION

Ease of adoption compared to high-level languages

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

Worth pondering: In the past 12 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.

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.

Needed: Memory-safety standardization

It's time to standardize principles and practices for software memory safety

Robert N. M. Watson^{*†}, John Baldwin^{††}, Tony Chen[‡], David Chisnall[§], Jessica Clarke[¶], Brooks Davis[¶], Nathaniel Wesley Filardo[¶], Brett Gutstein[¶], Graeme Jenkinson[¶], Ben Laurie^{*†}, Alfredo Mazzinghi[¶], Simon W. Moore^{††}, Peter G. Neumann[¶], Alex Richardson[¶], Alex Rebert[¶], Peter Sewell[¶], Laurence Tratt[¶], Murali Vijayaraghavan[¶], Hugo Vincent[¶], and Konrad Witaszczyk[¶]

^{*} University of Cambridge [†] Capabilities Limited [¶] SRI International
[#] Google, Inc [‡] Microsoft, Inc ^{**} SCI Semiconductor
^{††} Ararat River Consulting [§] Kings College London ^{##} Arm Limited

Draft version - 8 November 2024

Introduction	1
Background	2
Industrial best practices and market failure	5
Enabling business processes and market interventions	6
The memory-safety standardization gap	6
Audiences for memory-safety standardization	7
Goals for memory-safety standardization	8
Potential structures for one or more standards or documents	10
Adoption narratives and timelines	10
Potential events and interventions	11
Candidate timeline	12
Conclusion	14
Acknowledgements	14

Introduction

For at least two decades, endemic memory-safety vulnerabilities in software Trusted Computing Bases (TCBs) have enabled the spread of malware and devastating targeted attacks on critical infrastructure, national-security targets, companies, and individuals around the world. Over the last two years, the information-technology industry has seen increasing calls for the adoption of memory-safety technologies, framed as part of a broader initiative

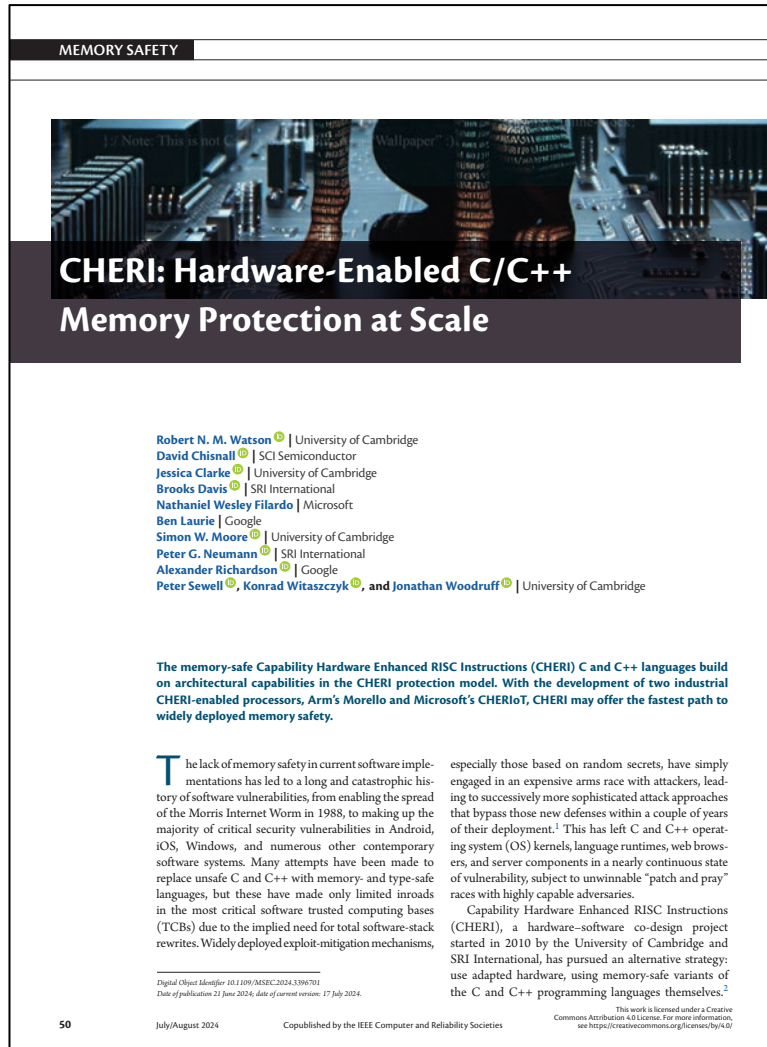
1

- As part of the policy dialog around memory safety, a key question:

“How can I ask for memory safety?”

- Procurement
- Regulation
- Contracts, liability, and insurance
- US National Academics of Sciences workshop in August 2024 engaged with this and other topics
- A whitepaper from Cambridge, SRI, Arm, Google, Microsoft, SCI, and others is being circulated; broader publication late this year
- Advocates a technology-neutral, vendor-neutral approach
 - Rust, CHERI, formal methods, compartmentalization, ...
- Aim to kick off a standardization effort next year

Learning more about CHERI



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

CHERI: Hardware-Enabled C/C++ Memory Protection at Scale

- See our technical reports for great detail:

Introduction to CHERI

CHERI C/C++ Programming Guide

CHERI ISA Specification

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

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

Conclusion

- New architectural primitives enable fine-grained C/C++ memory protection and scalable software compartmentalization
- Ideas portable across a range of architectures (Arm, RISC-V, ...) with full-scale software stacks running on them
- Prototype Arm Morello board shipped in 2022; 2.5 GHz high-performance prototype fabricated at 7nm
- Open-source Microsoft CHERI^{IoT} microcontroller released in 2023; to appear in FPGA and ASIC products over the next year
- Large and active community and software ecosystem!

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

