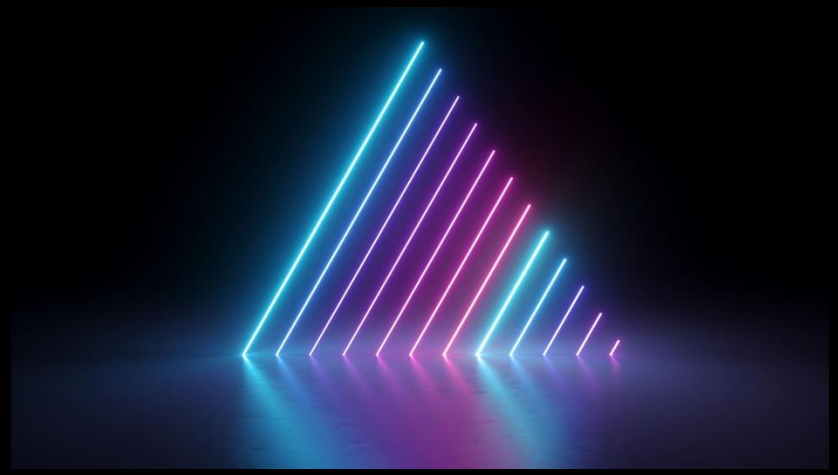
CHERI-seL4: Enhancing seL4's C/C++ userspace memory safety using CHERI



Hesham Almatary, Robert Watson, Capabilities Limited
seL4 Summit, 15 October 2024

CAPABILITIES LIMITED

Outline

Software Security

CHERI – Overview

- Memory Safety
- Software Compartmentalisation

CHERI-seL4

- Why?
- How?
- Progress
- Next steps

Software Security

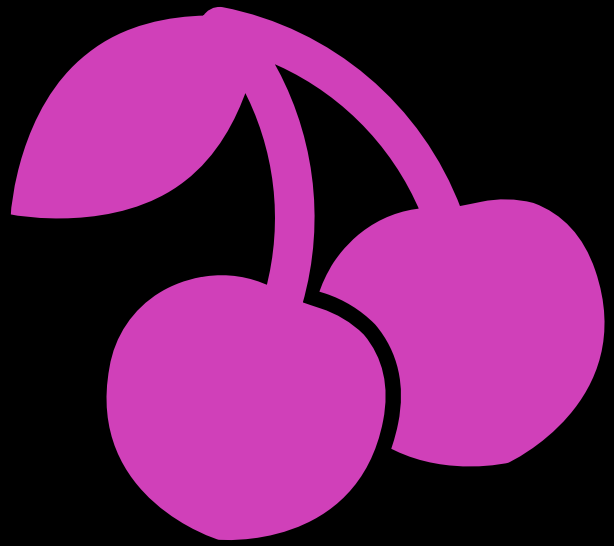
Memory Safety

70% of software vulnerabilities are memory-safety related in C/C++	e.g. Buffer overflows, control-flow, double-free, etc.	Enables confidentiality, integrity, and availability exploits

Software Compartmentalisation

Split up a large monolithic software into smaller compartments in order to reduce the attack surface and limit the effects of a successful attack only to the compromised compartment

What is CHERI?



- Hardware-software capability-based architecture
- Been under development and research since 2010 by University of Cambridge and SRI
- Prototypes on RISC-V and Arm (Morello)
  - Software ecosystem: LLVM, CheriBSD, Morello Linux, CheriFreeRTOS, CHERIoT etc.
- Core principles
  - Intentionality
  - Least privilege
  - Source-code compatibility

Cambridge/SRI	Arm	Microsoft	Google	Codasip
Piccolo, Flute, Toooba	Morello	CHERIoT	Ibex	Codasip 700

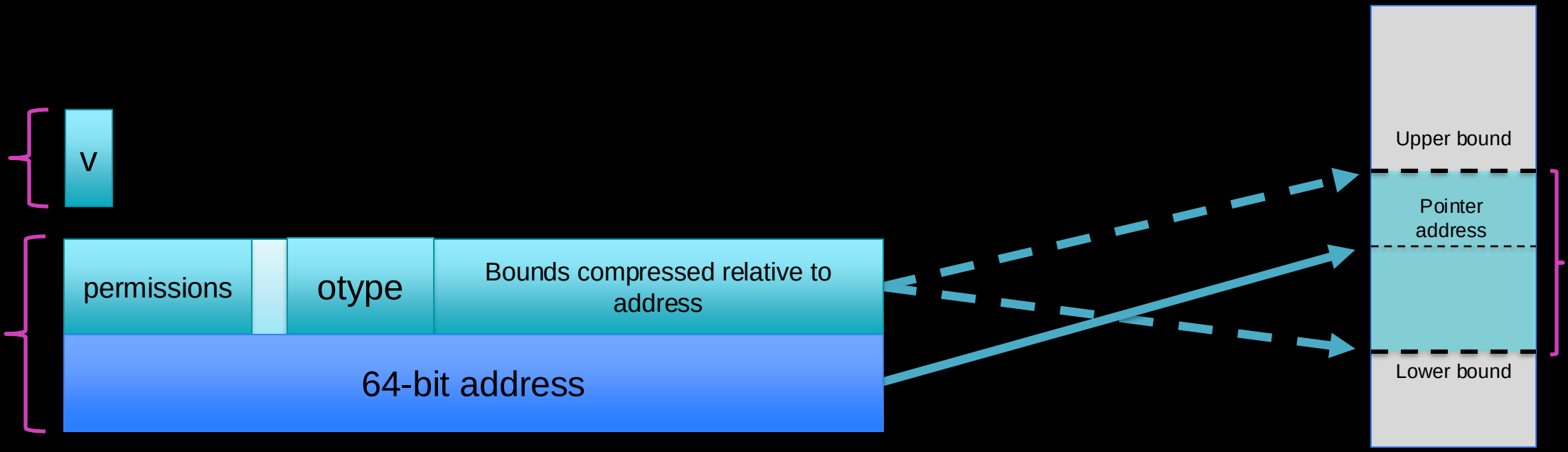
CAPABILITIES LIMITED

Starting point: CHERI on 64-bit systems

- Hardware knows about pointers
- Pointers carry bounds
- Pointers carry permissions
- Pointers can’t be created from thin air
- All guarantees are deterministic
- No guarantees rely on secrets

All memory access instructions require a valid pointer operand

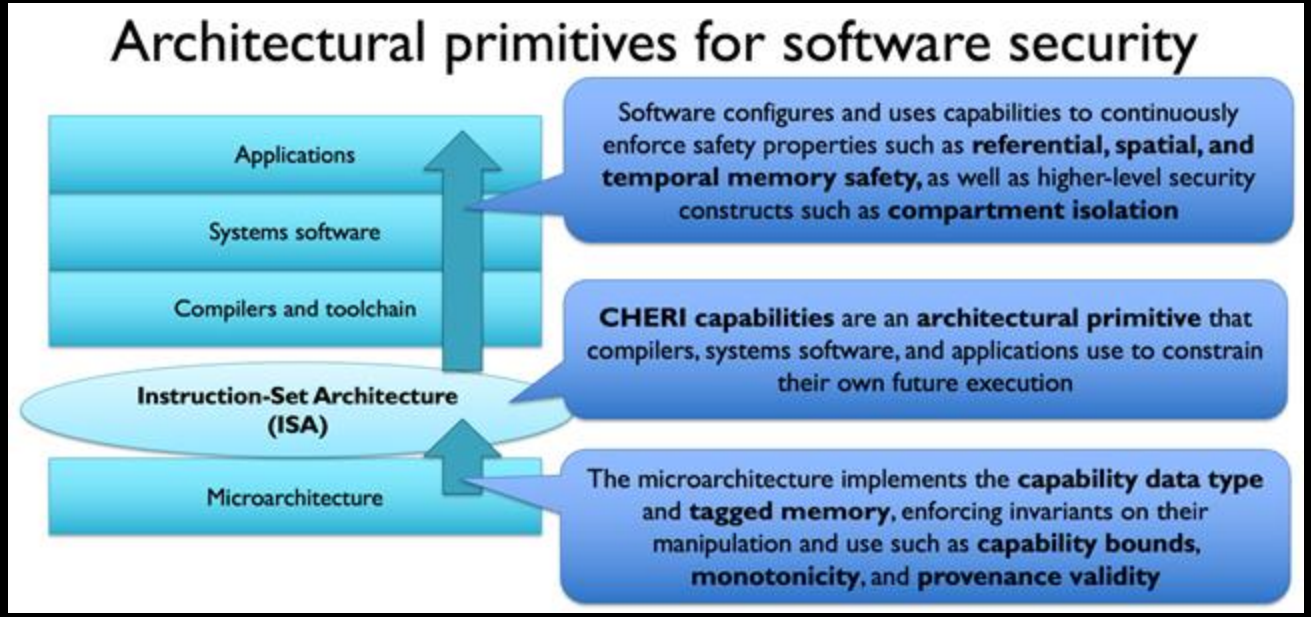
How does CHERI work



- 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

How does CHERI work

- Hardware:
  - Double register size
  - New CHERI instructions
  - Tagged memory
  - Hardware exceptions
- Software Protection Models:
  - Pointers Safety: hybrid or **purecap** (CHERI C)
  - Compartmentalisation

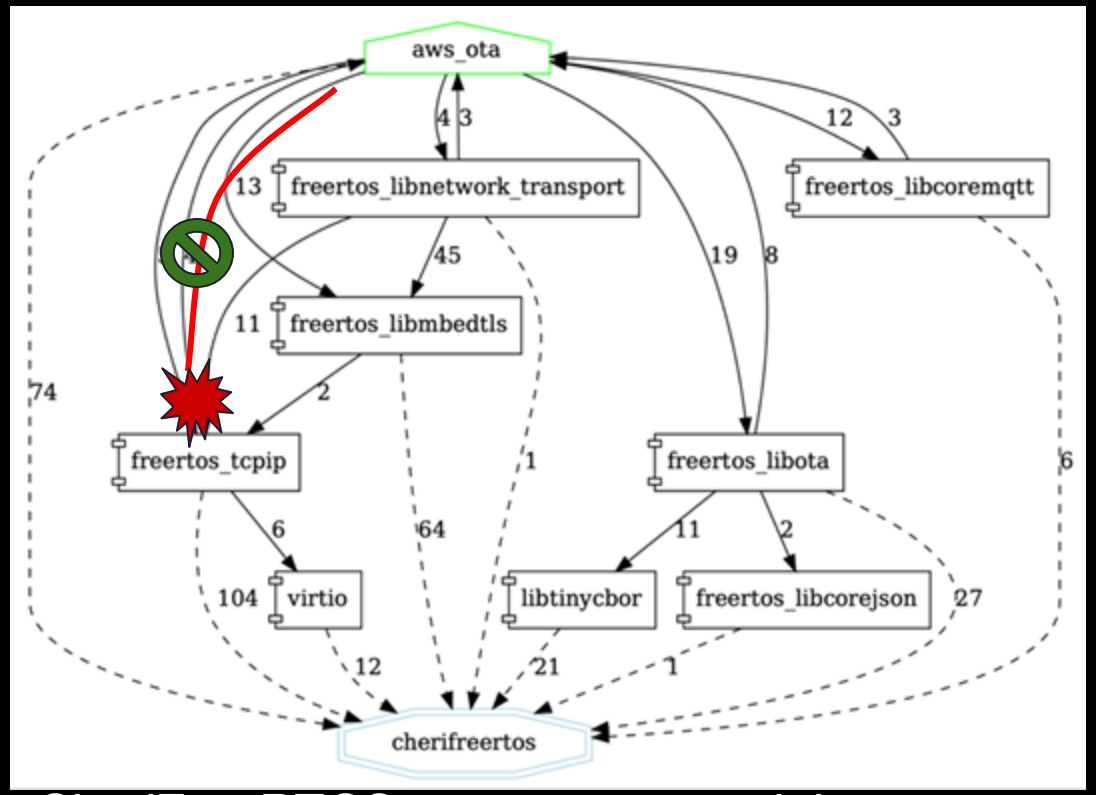


CHERI C Guidelines

- Pointers are unforgeable capabilities
- Pointers are not integers: ***sizeof(void *) != sizeof(long)***
- Minimum alignment for pointers is ***sizeof(void *)***
- **Provenance**: pointers are only derived from other pointers
  - ***uint32_t *mmio_region = (uint32_t *) (0xc0000000)*** - WRONG - Provenance violation
  - An OS or loader should create capabilities for MMIO regions
  - Code that performs bitwise arithmetic between ***uintptr_t*** is prone to error

    uintptr_t mutex_value = FLAG | (uintptr_t)(curthread) - WRONG - Provenance loss
    - If FLAG increases the bounds (e.g., embedding data in the high bits of the address)
  - Additional implications for code that makes assumptions about the shape of a pointer
- **Monotonicity**: A derived pointer cannot extend bounds or permissions
  - A memory allocator should set bounds on capabilities
- Further reading: CHERI C/C++ Programming Guide https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-947.pdf

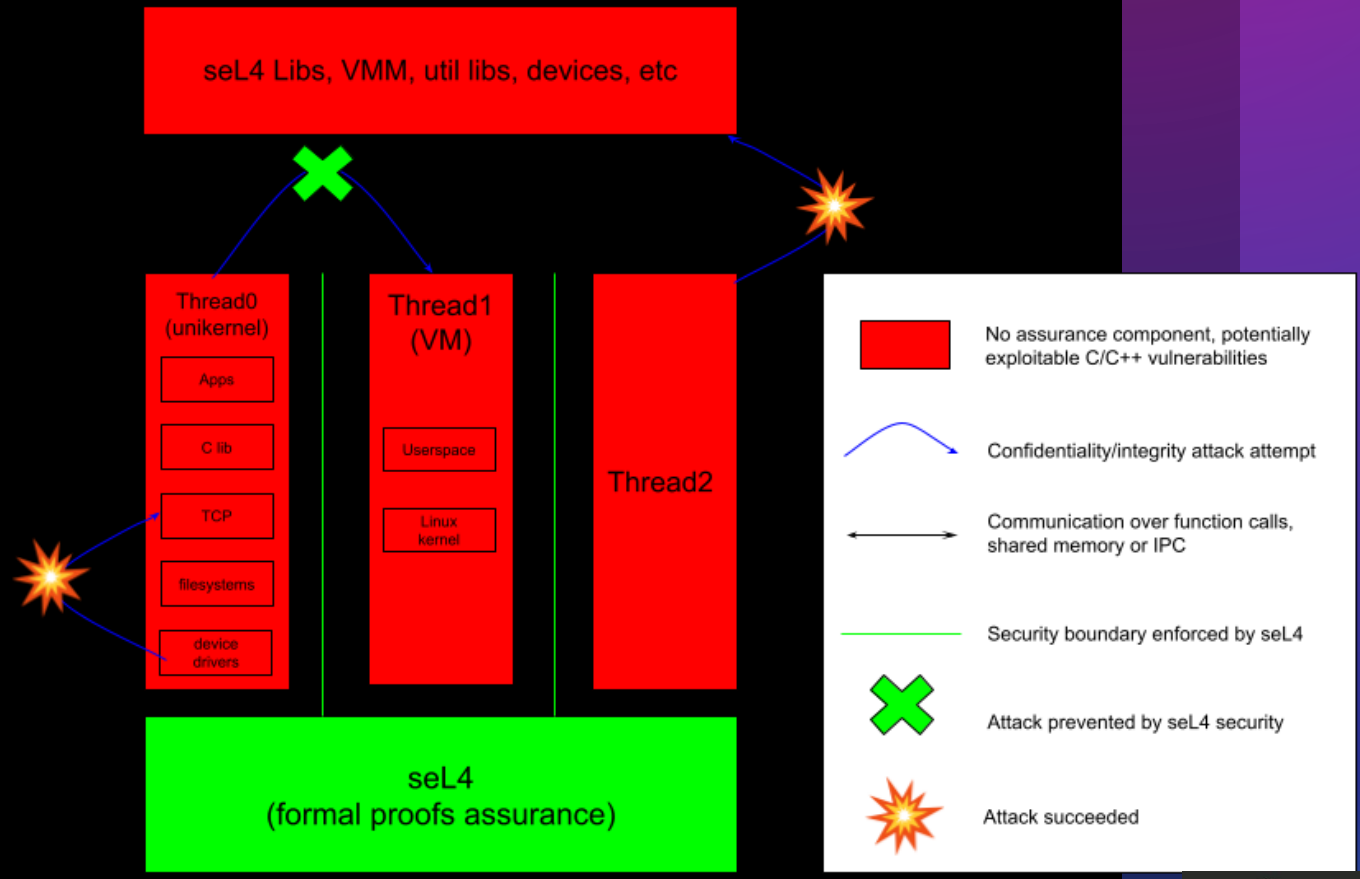
CHERI/CompartOS Software Compartmentalisation



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.

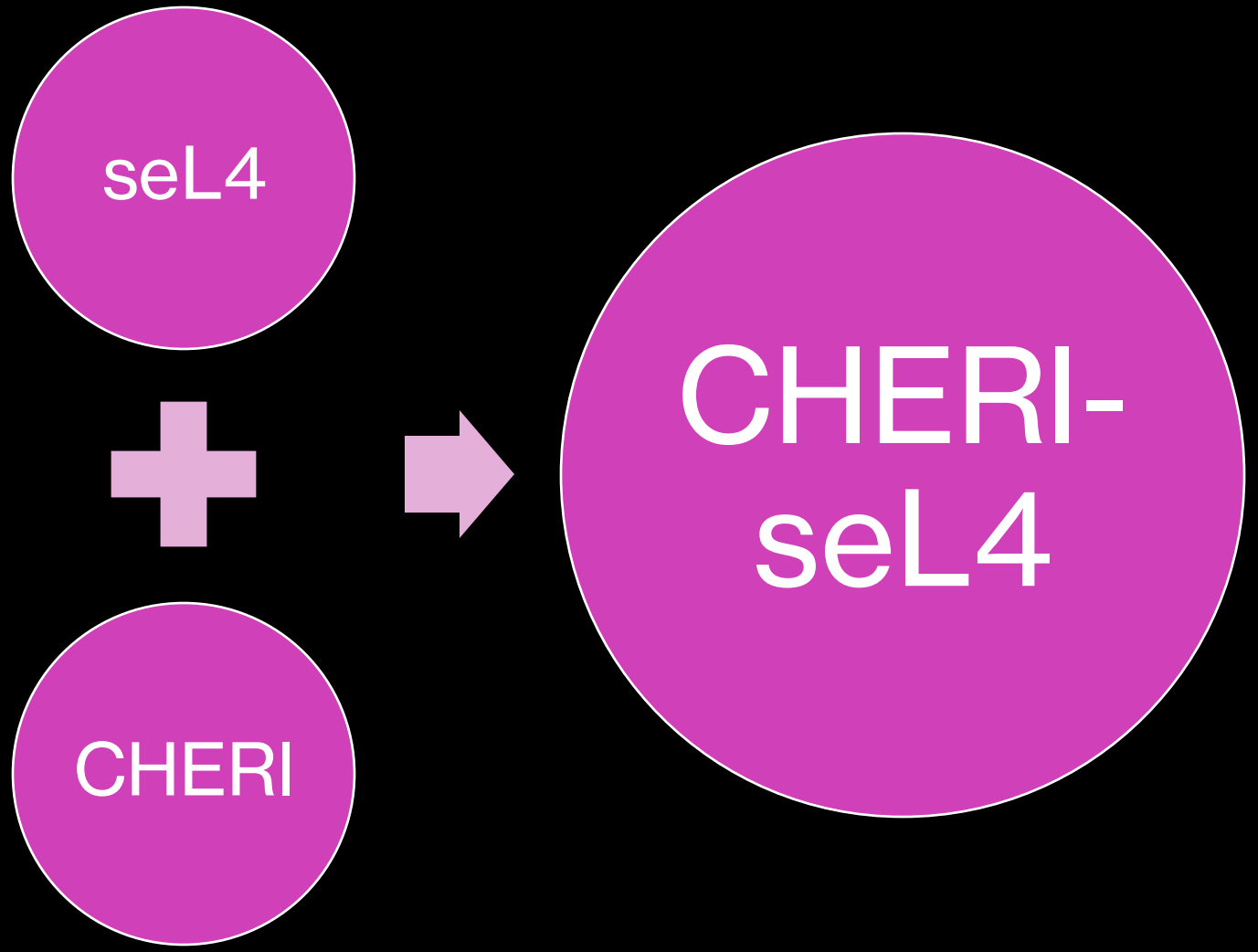
seL4

- seL4 is a secure microkernel – formally verified
- Gives isolation guarantees between user-level protection domains
- No C/C++ memory safety, guarantees at user-level



CHERI-seL4 – Why?

Great software technology with formal verification for separating inter-AS protection domains



Great hardware-software technology for single-AS memory safety and software compartmentalisation

What could it ***mean*** to bring both together?
This is an under-explored design space

CHERI-seL4 – Why?

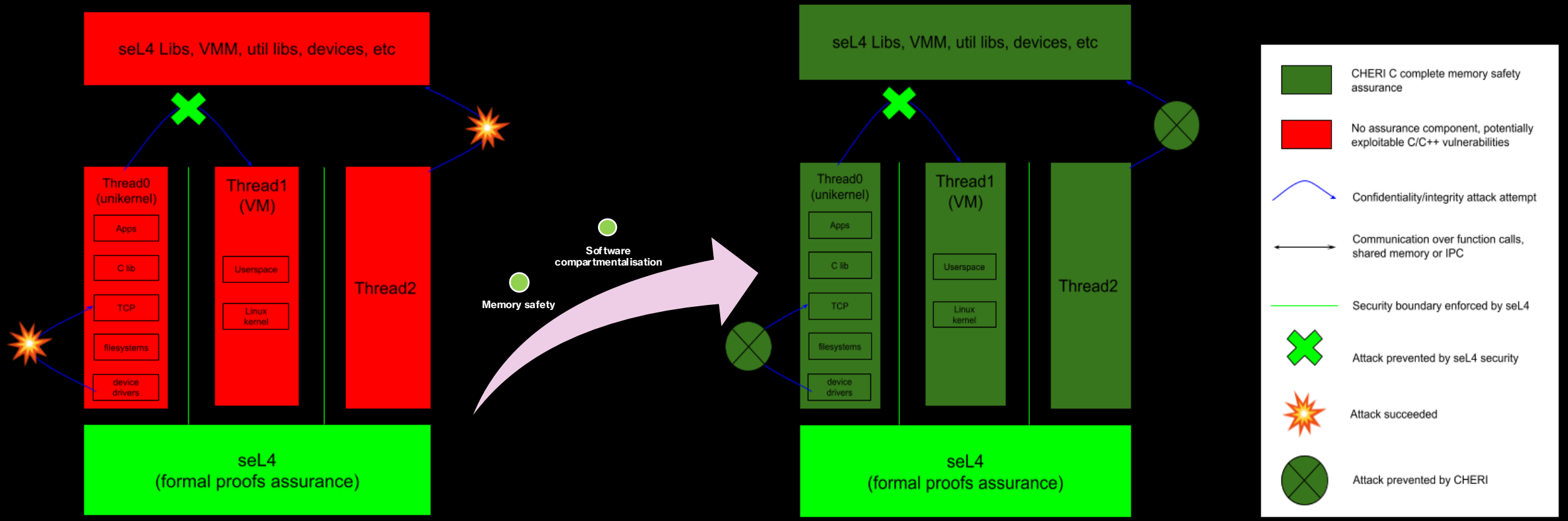
Combine two security technologies: seL4's ***formal verification*** for CIA isolation guarantees between seL4 protection domains and CHERI's ***memory-safety guarantees and software compartmentalisation*** within single-AS seL4 protection domains.

Enhances the overall system's security.

Reduces the overall attack surfaces that might get exploited

Protect lives, economies, infrastructures, sensitive customer data, etc

CHERI-seL4 – Why? Ideal goal



CHERI-seL4 – Approach and Goals

CHERI-seL4: shared unified kernel codebase, ideally no forks

CHERI OFF (=)

- Maintain all existing seL4 guarantees
- Minimal to no effects on verification
- No non-CHERI broken builds, tests, etc
- All CHERI-related code is hidden and guarded, and not compiled
- Just renaming, parameterisations, and clean-ups of shared code
- Same ABI, same performance, same types/sizes (eventually), same output binary

CHERI ON (+)

- Will not bypass any existing seL4 guarantees (eg. no privilege escalation)
- Not verified, but could be in the future
- Immediately enable *complete spatial memory-safety* (purecap) for C/C++ user-space.
- Feature-rich to allow design-space explorations on top (eg: ***software compartmentalisation, temporal safety***, etc)
- Same seL4 API, new CHERI purecap ABI

CAPABILITIES LIMITED

CHERI-seL4 – CHERI ON – Community Expectations

What are we trying to achieve?

- Design-space exploration
- System research
- Implement what we think is right, from CHERI perspective
- seL4 just provides the mechanism for the user to implement CHERI-based security policies, including memory-safety and software compartmentalisation
- Flexible, generic, and clean design and implementation to enable as many features as possible, use cases, and R&D projects on top

CHERI-seL4 – CHERI ON – How?

CHERI-seL4: user-space always purecap, but the kernel could be:

Hybrid

- Relatively minimum changes
- Would not change seL4's capability representation
- Likely to break verification, but probably not as much as purecap.
- Kernel pointers stay integers (*word_t*)

Shared

- libsel4's pointers to be capabilities
- IPC buffer messages to be of CHERI capabilities type
- System call arguments of CHERI capabilities type
- Register context saves/restores to be always CHERI-capability aware
- CHERI hardware initialisation at kernel boot
- CHERI initialisations for *BootInfo* and the root task, and any user pointers
- New CHERI fault message
- Export CHERI PTE bits to user pages

Purecap

- Maintain functional correctness and API capability of seL4 and its user-space
- Changes the seL4 capability representation to embed CHERI pointers.
- ***Any*** pointer is a CHERI capability
- Most kernel object sizes are increased to hold CHERI pointers
- Breaks verification
- Easier to port and reason about (from CHERI perspective).
- Provenance: no need to re-construct CHERI capabilities in the kernel after booting to user, except in very rare circumstances

CHERI-seL4 – Progress

We have complete all-purecap sel4test running and passing all tests

Firmware

- Elfloader: Could optionally be built in purecap for some targets
- OpenSBI: Not purecap yet, but hybrid just to preserve tags and capabilities

Kernel

- PR and RFC submitted to support CHERI-seL4 in the kernel
- CHERI-RISC-V (RV32 and RV64)
- Arm Morello: 4 new platforms submitted – QEMU, FVP, bhyve, and Morello SoC board.
- Purecap kernel only so far, with shared snippets for hybrid
- Enables building and running complete all-purecap seL4 user-space

Userspace

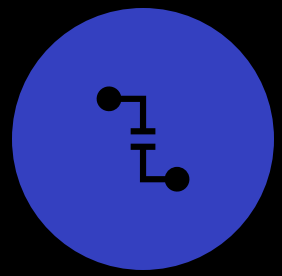
- Able to run complete purecap projects (sel4test and sel4bench)
- All C/C++ seL4 user libraries that sel4test and sel4bench are using are ported to purecap
- sel4test passes all tests in purecap

```
        Is connection handle valid?
Test SERSERV_PARENT_010 passed
Starting test 93: SYNC001
Starting test 94: SYNC002
Starting test 95: SYNC003
Starting test 96: SYNC004
Starting test 97: THREADS0004
Starting test 98: THREADS0005
Starting test 99: TLS0001
Starting test 100: TLS0002
Starting test 101: TLS0006
Starting test 102: TRIVIAL0000
Starting test 103: TRIVIAL0001
Starting test 104: TRIVIAL0002
Starting test 105: VSPACE0000
Starting test 106: VSPACE0001
Starting test 107: VSPACE0002
Starting test 108: VSPACE0003
Starting test 109: VSPACE0004
Starting test 110: VSPACE0005
Starting test 111: VSPACE0006
Starting test 113: Test all tests ran
Test suite passed. 113 tests passed. 56 tests disabled.
All is well in the CHERI-seL4 universe
```

CHERI-seL4 – Remaining work

HYBRID KERNEL

BENCHMARKING



RELEASING PURECAP USER-SPACE AND FIRMWARE

DOCUMENTATION AND TUTORIALS

CHERI-seL4 – Future work

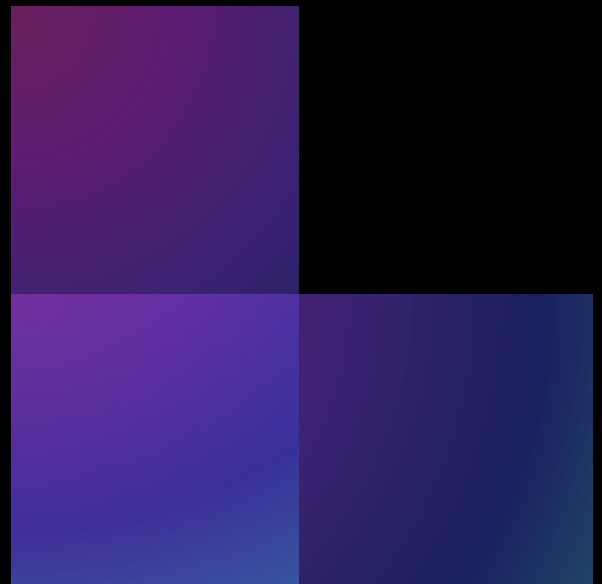
- Hypervisor support
- MCS and SMP support
- CHERI's temporal safety
- Software Compartmentalisation (e.g., to automatically sandbox third-party libraries)
- Port other C/C++ projects to CHERI-seL4 (e.g., Microkit, LionsOS, Unikernels)
- Lots of other R&D project ideas

Conclusions

CHERI-seL4 is aiming to combine both seL4 and CHERI technologies to enhance the overall security

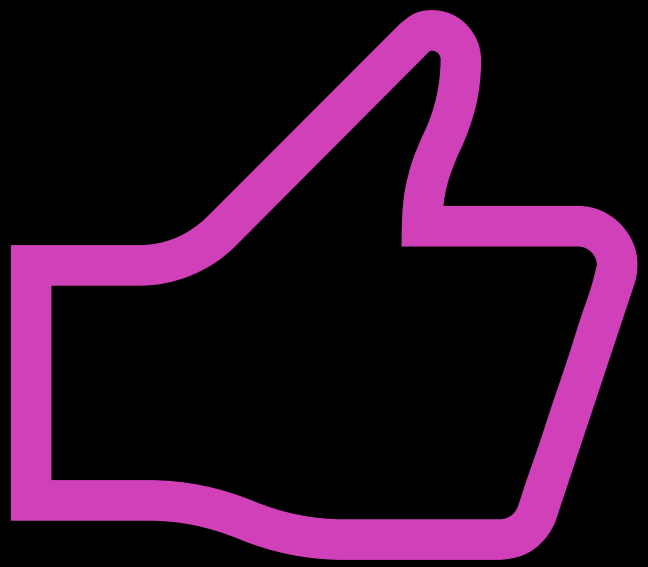
Good progress on CHERI-seL4 shows it is feasible and opens lots of future potentials

Still work-in-progress. We would love to get your input. Help us make this the most useful for the seL4 ***and*** CHERI communities



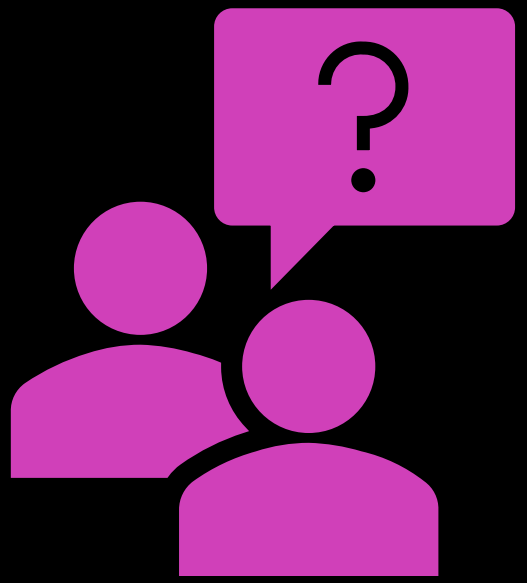
CAPABILITIES LIMITED

Acknowledgment



- Thanks to the CHERI group and David Chisnall for permitting usage of some of their slides
- We also thank TrustedST, MCA, Sid Agrawal, Codasip, and seL4 foundation for the discussions and collaboration on this work so far.

CAPABILITIES LIMITED



heshamalmatary@capabilitieslimited.co.uk

CAPABILITIES LIMITED

Resources

- Watson, Robert NM, et al. Capability hardware enhanced RISC instructions: CHERI instruction-set architecture (version 9). No. UCAM-CL-TR-987. University of Cambridge, Computer Laboratory, 2023.
- Almatary, Hesham, et al. "CompartOS: CHERI compartmentalization for embedded systems." arXiv preprint arXiv:2206.02852 (2022).
- Almatary, Hesham. CHERI compartmentalisation for embedded systems. Diss. 2022.
- Almatary, Hesham, Alfredo Mazzinghi, and Robert NM Watson. "Case Study: Securing MMU-less Linux Using CHERI." SE 2024-Companion. Gesellschaft für Informatik eV, 2024.
- CHERI Website: https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/

Why not just use Rust userspace on seL4

- Unsafe pointers
- Toolchain supply chain
- No dynamic software compartmentalisation that defends against unknown zero-day vulnerabilities.
- Maintenance and cost overhead to re-write existing C/C++ applications in Rust
- Re-writing codebase in Rust could introduce new bugs
- Re-writing third-party libraries (eg crypto) in Rust is hard
- Cannot compartmentalise existing C/C++ libraries automatically like with CHERI
- Learning overhead for developers because Rust is relatively new
- Ideally port Rust to run on CHERI
- CHERI Myths: I don't need CHERI if I have safe languages

CHERI-seL4: Current progress in a figure