Memory Safety in Telecommunications with CHERI?

Merve Gülmez

Experienced Researcher
Ericsson Research Security

Thomas Nyman

Expert Trusted HW & SW Technologies
Ericsson Product Security

Motivation: Memory Safety

C and C++ still preferred languages for **systems programming**, e.g., OS kernels, core libraries etc.

- Almost all other systems not written in C or C++ call routines written in C or C++

Majority of vulnerabilities written in C and C++ are "**memory-safety**" vulnerabilities

Routinely exploited by malicious threat actors

- ≈70% of "zero-day*" exploits memory-safety vulnerabilities [1]

*) vulnerabilities that threat actors know about, and for which there are no patches available from the software vendor

[1] Google Project Zero (2025). 0day "In the Wild".

Memory Safety Hardening Challenge

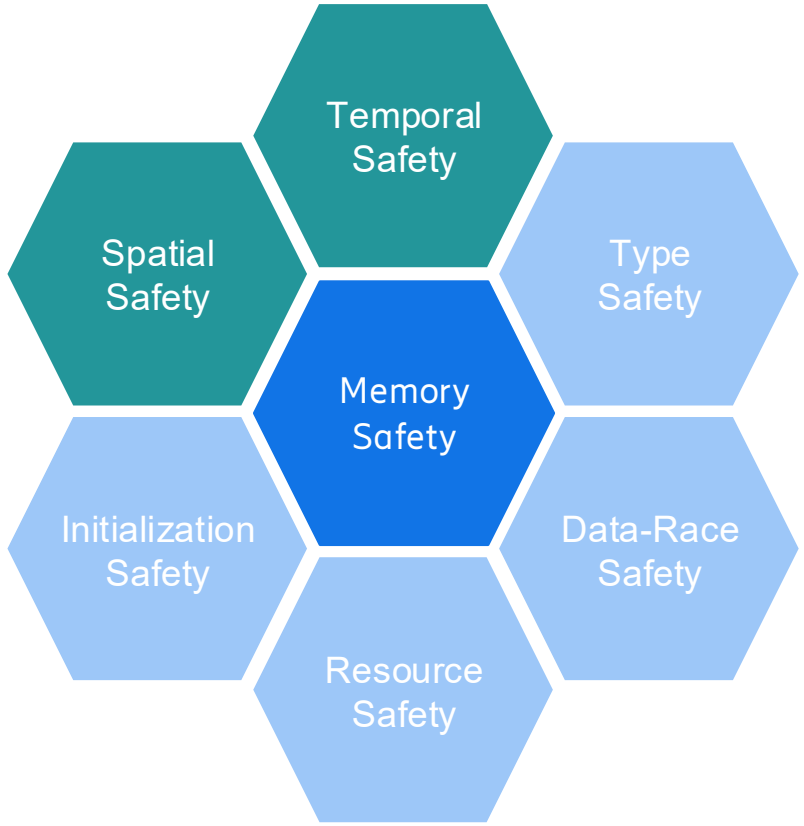
Concerns around **memory safety** in software industry are increasing thanks to regulatory attention

- Memory safety is a combination of **correctness** aspects
- Memory safety principles being standardization in ETSI

We can harden software against memory-safety bugs by

- **Reducing their prevalence:** use memory-safe programming languages, e.g., Rust, or static and dynamic analysis tools
- **Reducing their impact:** use run-time defenses, e.g., stack canaries or memory-safety enforcing hardware, e.g., CHERI (Capability Hardware Enhanced RISC Instructions [2])

Memory-safe hardware particularly attractive for existing codebases in unsafe languages, such as C and C++



[2] J. Woodruff et al., *The CHERI capability model: Revisiting RISC in an age of risk*, ISCA '14

Why are regulators interested in memory safety?

ERICSSON

Cybersecurity authorities' chosen focus to achieve more secure software by systematically reducing vulnerabilities

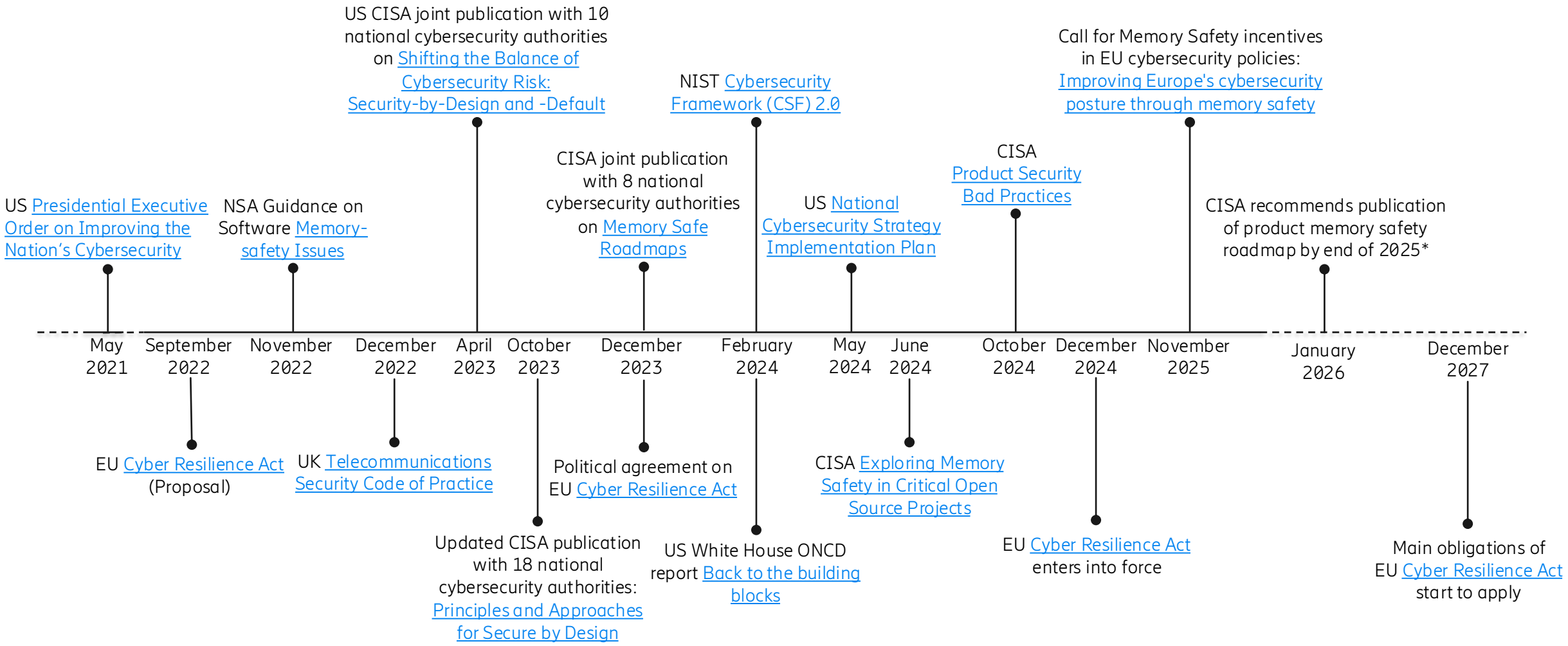
Major cause for the release of urgent security updates and need for timely patching

Regulators perceive significant resources spent responding to vulnerabilities through onerous patch management programs and incident response activities

National security priority for US; White House ONCD*, CISA†, and NSA‡ recommend memory-safe programming languages and "secure by design" practices

*) Office of the National Cyber Director
†) Cybersecurity and Infrastructure Security Agency
‡) National Security Agency

Recent regulatory attention

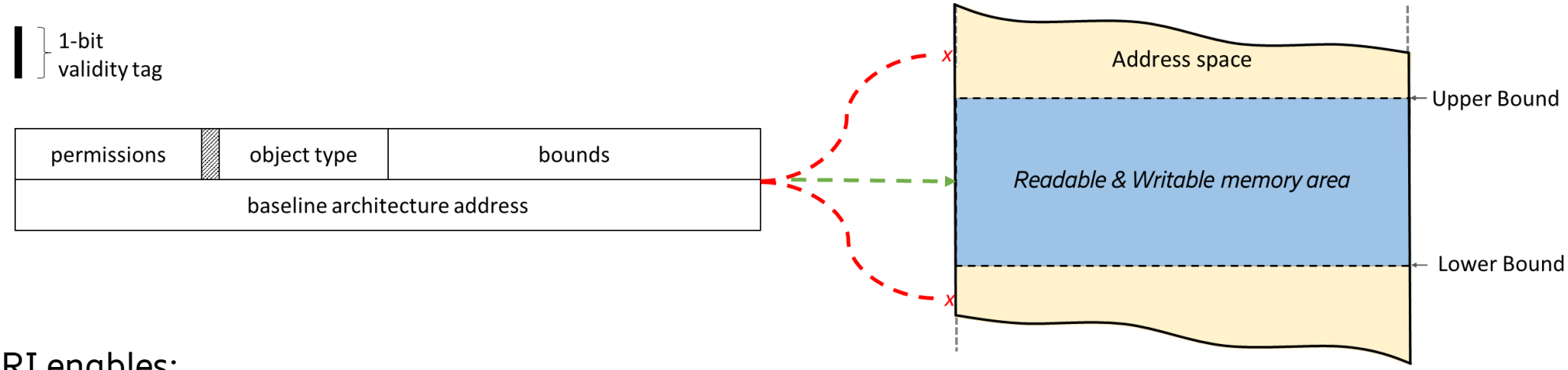


*) Not expected for products with EOL before 2030

CHERI: Capability Hardware Enhanced RISC

CHERI is a **processor architectural protection model** developed by University of Cambridge and SRI International

- Revisits **capability-based addressing** from late 1950's and 60's mainframes
- Replaces conventional memory pointers with **integrity-protected capabilities**



CHERI enables:

- **Fine-grained memory protection:** deterministically prevents conditions necessary for buffer overflows, control-flow hijacking, use-after-free (with CHERI-aware allocator), and other memory attacks
- **Scalable compartmentalization:** *CHERI-processes"* to co-locate within same virtual address space

CHERI Adoption Prerequisites

CHERI-enabled processor—no off-the-shelf production silicon currently available

- Evaluation platform: **Arm Morello** system-on-chip SoC (prototype, based on Neoverse N1 / A76 microarchitecture)
- FPGA-based options available for research: **CHERI-RISCV softcores** (open-source, suitable for experimentation)

→ Future production adoption depends on chip vendors integrating CHERI into mainstream silicon

CHERI-aware operating system—off-the-shelf OSs will not leverage CHERI protections

- **CheriBSD** (FreeBSD-based) primary reference OS for Morello, CHERI-aware Linux developed by Linaro
- Full **CHERI software stack** is needed: OS kernel, system libraries, and runtime support (e.g., memory allocator)

CHERI-enabled compiler e.g., CHERI-LLVM / Clang or Rust compiler with capability support

- Standard compilers (GCC, mainstream Clang) do not generate capability-aware code
- Toolchain must understand and emit **capability-tagged pointers** instead of conventional pointers

→ CHERI-LLVM and `rustc` prototypes available, upstreaming depends on CHERI standardization

CHERI-enabled applications—porting effort due to:

- **Pointer arithmetic** that violates bounds assumptions
- **Type punning** and unsafe casting patterns
- **Third-party dependencies** lacking CHERI support

→ As CHERI ecosystem matures, dependency-related friction is expected to decrease

Is CHERI viable for telecommunications systems?

Ericsson ported 5G RAN (RLC/MAC) benchmarks written in C/C++ to **Arm Morello** prototype SoC [3]

- Only ~**1% of source code** required changes for CHERI compatibility (if supported by OS and toolchain)
- Performance overhead observed, but largely tied to Morello prototype design constraints
- Cambridge research suggests optimized implementations could improve performance
  - improve speculation on capabilities
- Arm Morello program concluded in 2024
  - no new Arm-based demonstrators expected

[3] F. Hessner, *Memory-Safe 5G Software Using the CHERI Hardware Architecture*, MSc Thesis 2024

Outlook for CHERI on RISC-V

CHERI concepts **being standardized** in RISC-V:

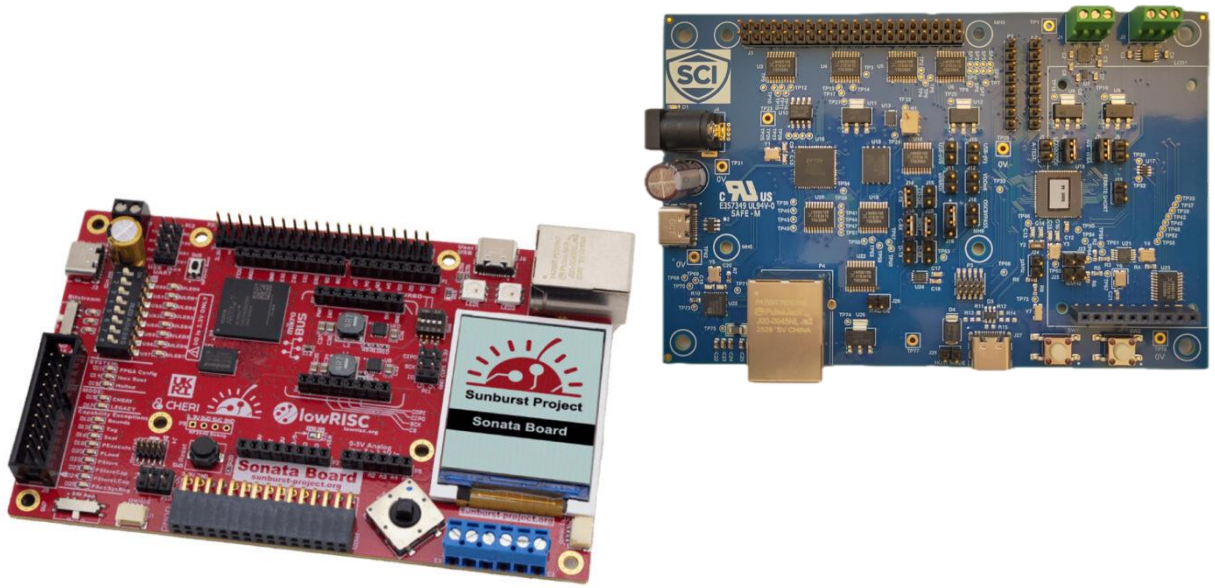
- Compressed capability format (initially 64-bit profile)
- Capabilities for spatial memory safety
- Capabilities for temporal safety (use-after-reallocation)
- Capabilities for compartmentalization

Different CHERI implementations add **own extensions**:

- Example: CHERIoT platform and RTOS add "non-standard" **temporal-safety acceleration** for resource-constrained embedded IoT device

Multiple **industrial implementations**:

- Microsoft's 32-bit CHERIoT RISC-V Ibex and Kudu
- Codasip's 64-bit CHERI RISC-V IP
- SCI's ICENI first silicon based on CHERIoT Ibex



Research challenges and ongoing investigation

ERICSSON

Considerations for CHERI application processors:

- Improve speed and precision of temporal safety [4]
- Safe speculation without sacrificing performance [5]

New compartmentalization models:

- Linkage and library-based compartmentalization
- CHERI-based TEEs [6] and attestation [7]

Security evaluation tooling:

- Correct choice of compartment boundaries?

Use of CHERI in memory-safe languages:

- Protecting or compartmentalizing unsafe Rust and mixed-language applications

New security properties:

- Additional memory safety properties [8]
- Properties beyond memory safety [9]

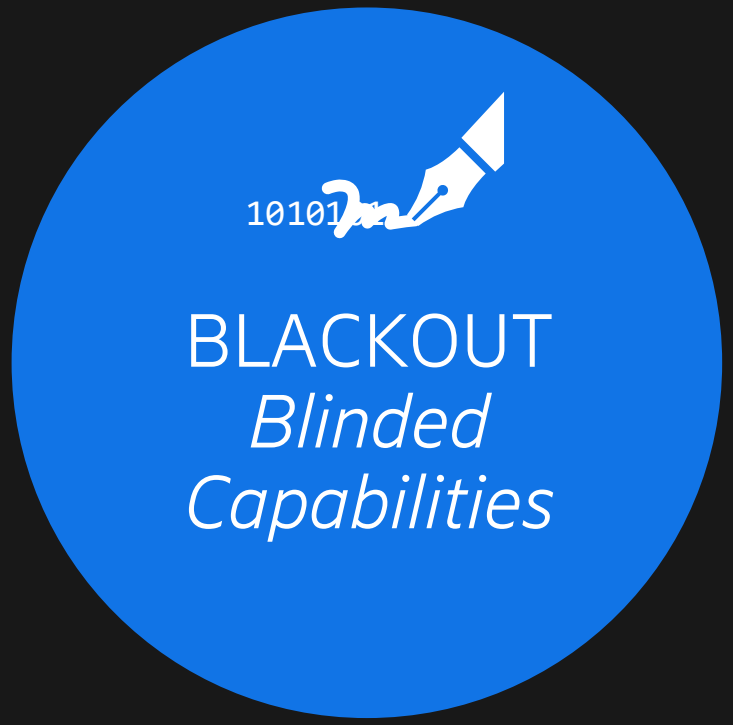
Single address-space operating systems

- using solely CHERI for isolation (no-MMU)

[4] M. Gülmez et al., *PICASSO: Scaling CHERI Use-After-Free Protection to Millions of Allocations using Colored Capabilities*, arXiv:2602.09131 [cs.CR]
[5] F. Fuchs et al., *Safe Speculation for CHERI*, ICCD '24
[6] T. Strydonck., *CHERI-TrEE: Flexible enclaves on capability machines*, EuroS&P '23
[7] J. Rousseau, *Cerisier: A Program Logic for Attestation in a Capability Machine*, arXiv:2604.13638v2 [cs.PL]
[8] M. Gülmez et al., *Mon CHERI: Mitigating Uninitialized Memory Access with Conditional Capabilities*, S&P '25
[9] H. ElAtali et al., *BLACKOUT: Data-Oblivious Computation with Blinded Capabilities*, CCS '25

Ericsson Joint CHERI Work With Academia

Better memory-safety coverage



Security beyond memory safety

More accurate and faster temporal-safety

Takeaways

CHERI is a **promising but not yet production-ready** technology for telecommunications systems

- Realistic demonstrators (e.g., Arm Morello) are essential for industry evaluation
- Broad ecosystem support—chip vendors, open-source community—required for adoption

Exciting **research opportunities** enabled by CHERI and its pre-commercialization

- Fast temporal safety in application processors for heap, stack, and compartments
- Relevance of CHERI for Rust and mixed-language applications

Memory Safety in Telecommunications with CHERI Blogpost

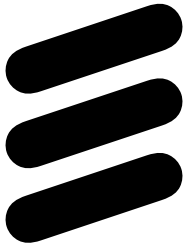
https://www.ericsson.com/en/blog/2024/9/memory-safety-in-telecommunications-with-cheri

BLACKOUT

https://dl.acm.org/doi/10.1145/3719027.3765169

PICASSO

https://arxiv.org/abs/2602.09131



ERICSSON

https://www.ericsson.com/en/security