

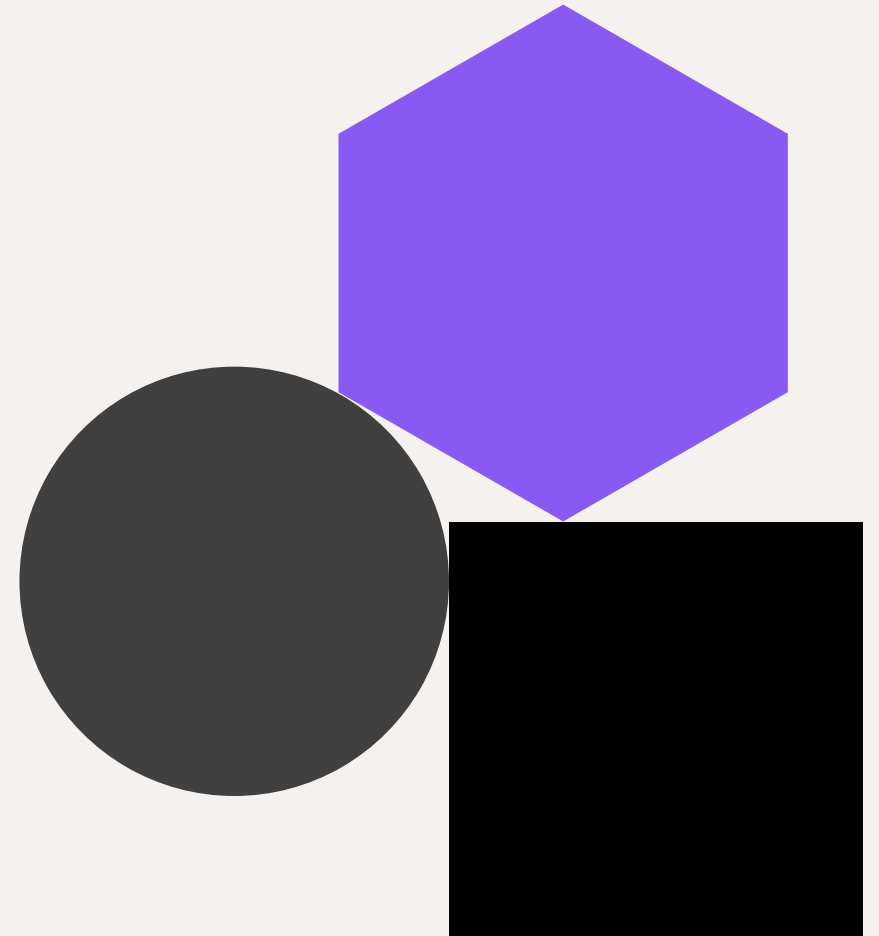


CHERI and Linux : Back to the Future

Integer pointers? Where we're going we don't need integer pointers.

Carl Shaw

August 2025



→ Agenda

- Back: a brief history of Capabilities
- Introducing CHERI
- CHERI and Linux
- The future



→ A brief history of Capabilities

CHERI:

Capability **H**ardware **E**nhanced **RISC** Instructions

Extends conventional ISAs to support a new hardware-supported *architectural capability* data type

→ A brief history of Capabilities

Programming Semantics for Multiprogrammed Computations

Jack B. Dennis and Earl C. Van Horn

Massachusetts Institute of Technology, Cambridge, Massachusetts

Communications of the ACM

Volume 9 / Number 3 / March, 1966

"Protection. In an MCS [multiprogrammed computer system], a computation must be denied access to memory words and other objects of computation unless access is authorized"

<https://dl.acm.org/doi/10.1145/365230.365252>

→ A brief history of capabilities

"Each capability ... locates by means of a pointer some computing object, and indicates the actions that the computation may perform with respect to that object."

Capabilities for memory contained *access indicators*:

X	executable as procedure including internal read references for constants.
R	readable as data but not executable.
XR	executable as procedure and readable as data.
RW	readable and writeable as data.
XRW	executable as procedure and readable and writeable as data.

(Object) Capability:

An unforgeable, transferable token that authorizes the use of an object.

It combines (object name, access rights)

→ A brief history of capabilities

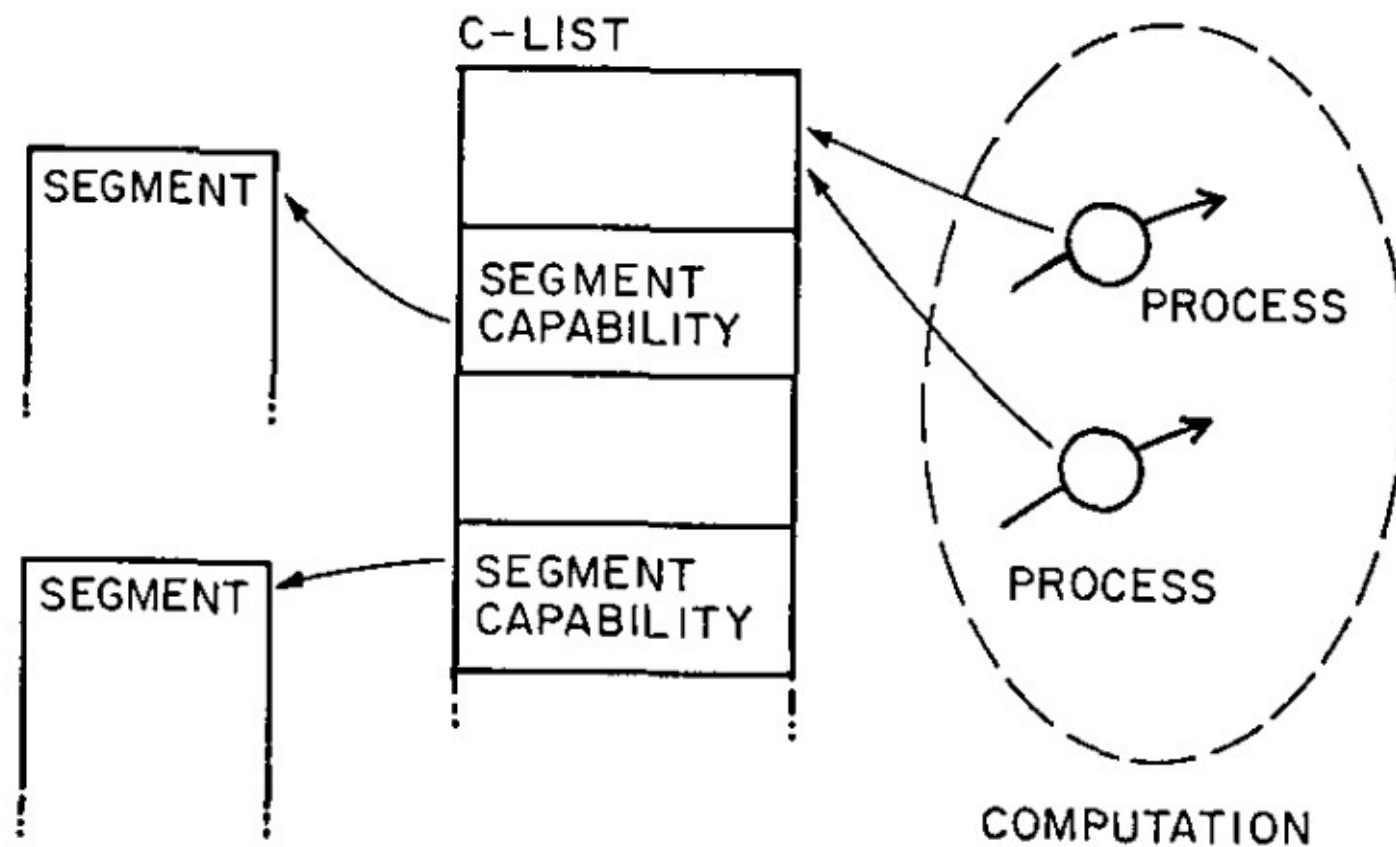


FIG. 1. A computation

→ A brief history of Capabilities

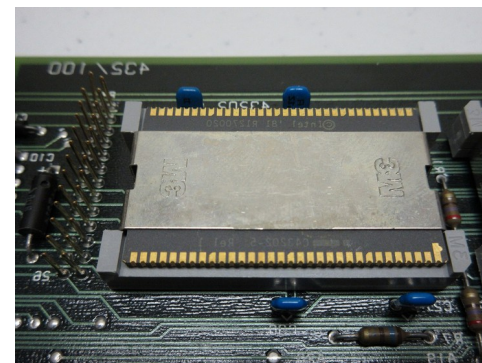


By author

**University of
Cambridge CAP
computer**
(research)
1970-1977



**Plessey System
250 (PP250)**
1972-1990s



By _brouhaha_ - Intel C43202 chip, CC BY-SA 2.0,
<https://commons.wikimedia.org/w/index.php?curid=113440105>

Intel iAPX 432
1975-1986



Arm Ltd.

Arm Morello
(CHERI research)
2022

→ A brief history of Capabilities

KeyKOS

OS for IBM S/370
1975-1992

EROS

Extremely Reliable
Operating System
(research)
1991-2005

CapROS

Capability-based
Reliable Operating
System
(research)
2005-2022

seL4

2009-

→ A brief history of Capabilities

Capabilities in Linux:

- Socket descriptors
- File descriptors
- Kernel capabilities
- Page Table Entries!

→ Introducing CHERI

→ Introducing CHERI

- A new approach to scalable, deterministic, capability-based addressing (no C-lists)
- Openly extends *existing* ISAs
- Hybrid Capability Architecture: backwards compatible and works with MMUs, Hypervisors and existing programming languages
- 15 years of development (so far!)
- First developed by SRI International and University of Cambridge

SRI International



UNIVERSITY OF
CAMBRIDGE



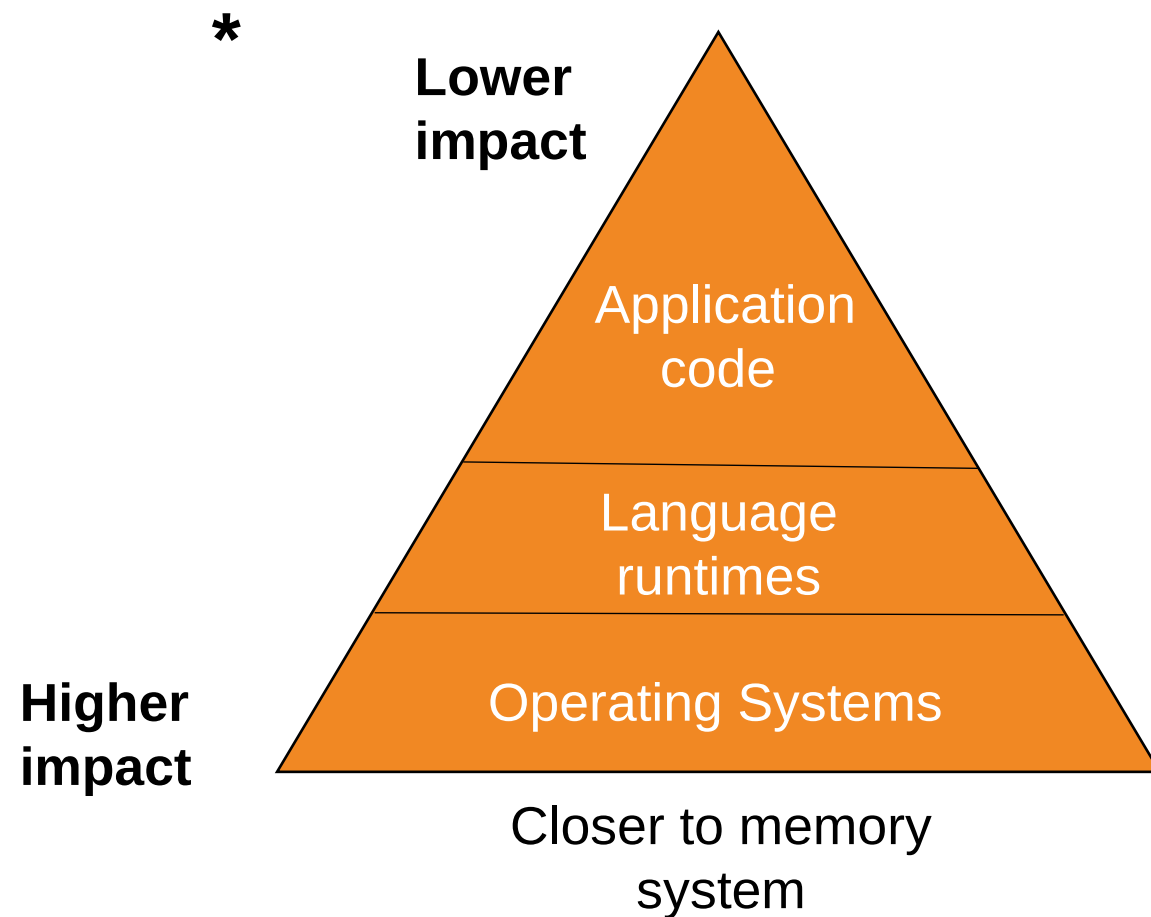
CHERI

CHERI Alliance – Industry-led
security technology

→ Introducing CHERI

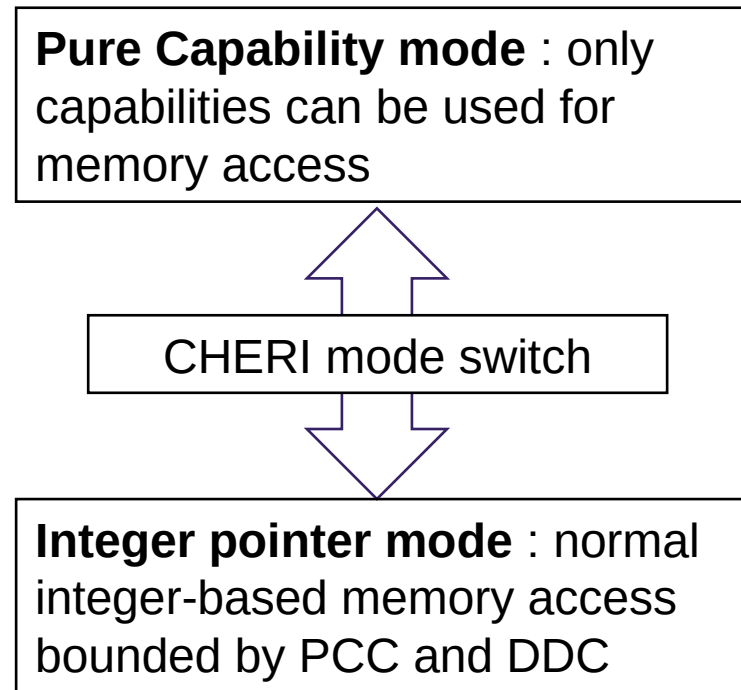
Design goals:

- Provide hardware-enforced memory safety
 - C/C++/Rust can all benefit
 - Minimal, if any, code rewriting*
- Provide fine-grained, efficient compartmentalization



→ Introducing CHERI

- The ISA is extended with instructions to create and modify capabilities
- The architecture is extended to add hardware capability support
 - Registers are extended to hold capabilities
 - Capabilities can be loaded and stored to memory
- No C-List indirection : pointers are replaced in software by capabilities (in pure capability code)
 - Memory pointers are no longer just integers, they are (address, access rights)
- ALL loads and stores to memory are checked against an authorizing capability
 - The Program Counter is a capability (PCC)
 - We have a Default Data Capability (DDC)



→ Introducing CHERI

- Capability format:



A capability on a 64-bit CHERI-RISC-V CPU

"It is predicted that the use of tags to indicate capabilities will dominate."

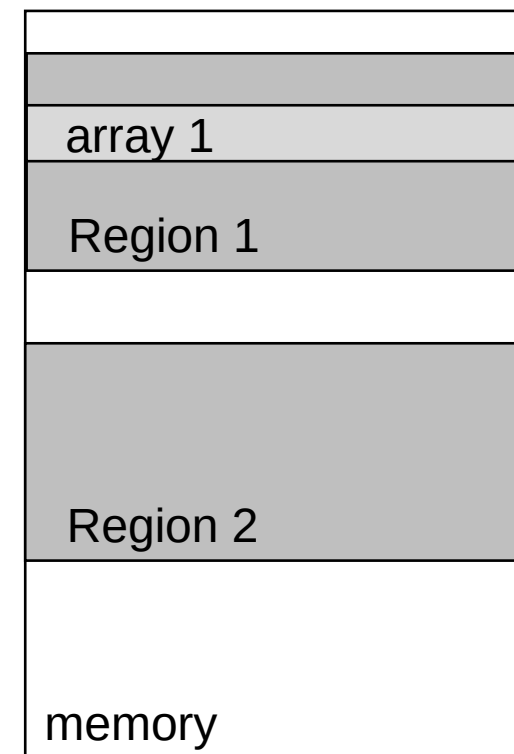
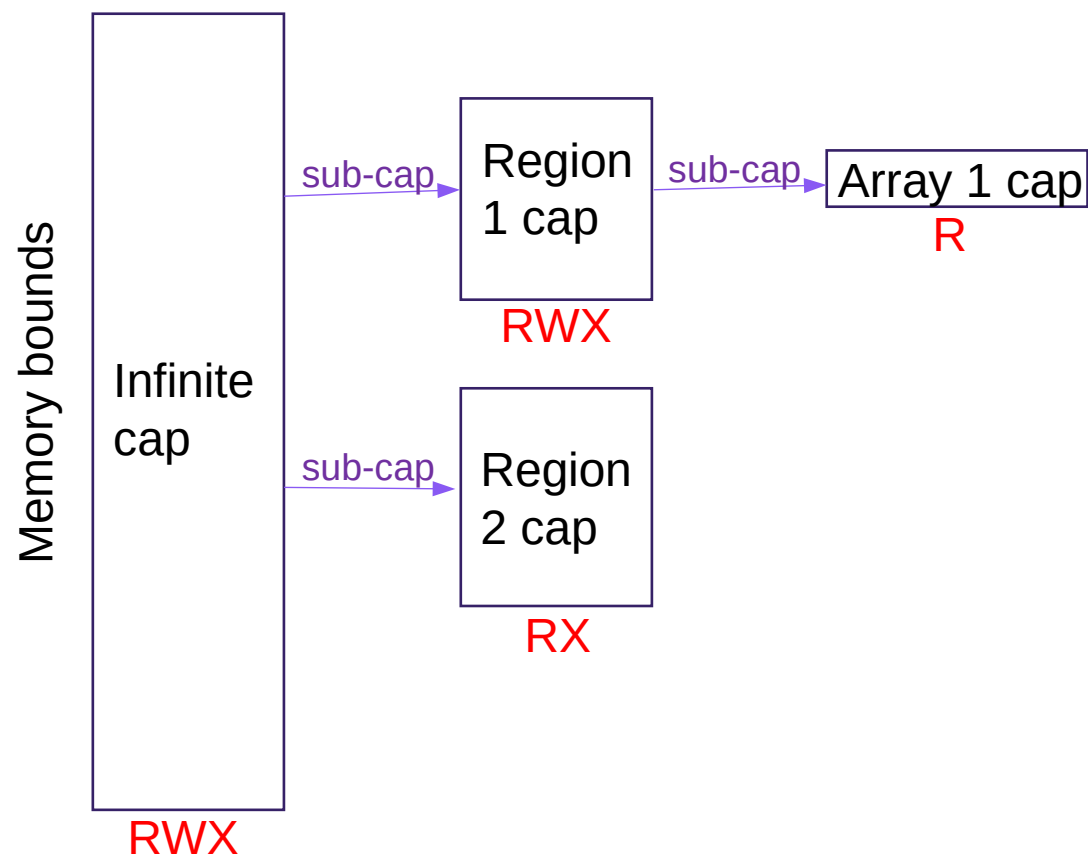
R.S. Fabry, "Capability-based Addressing", Communications of the ACM, Volume 17, Number 7, 1974

→ Introducing CHERI

Architectural rules for Capability use:

- **Provenance:** CHERI instructions can only be used to construct valid capabilities explicitly from other valid capabilities.
 - This property applies to capabilities in registers and in memory.
- **Monotonicity:** When an instruction constructs a new capability, the resulting capability cannot exceed the permissions and bounds of the capability from which it was derived.
 - Rights are always attenuated.
- **Reachable capability monotonicity:** on execution of arbitrary code, the *reachable capabilities* (in registers, memory and by creation of capabilities) cannot increase until execution is yielded to another domain.

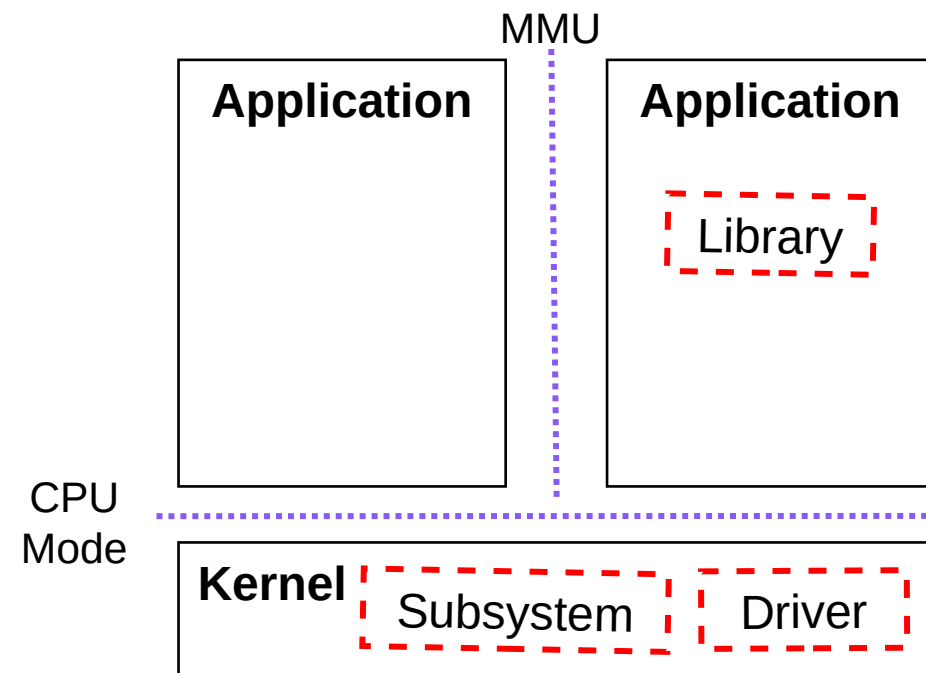
→ Introducing CHERI



→ CHERI and Linux

→ CHERI and Linux

- It provides strong, hardware-enforced runtime memory safety
 - Critical for C / C++
 - Compiler does most of the heavy lifting
- It allows us to implement least privilege software compartmentalization
 - Untrusted software components isolated
 - Reduces supply chain attack
 - Allows for safer mixed-language applications (e.g. Rust or Python calling C libraries)

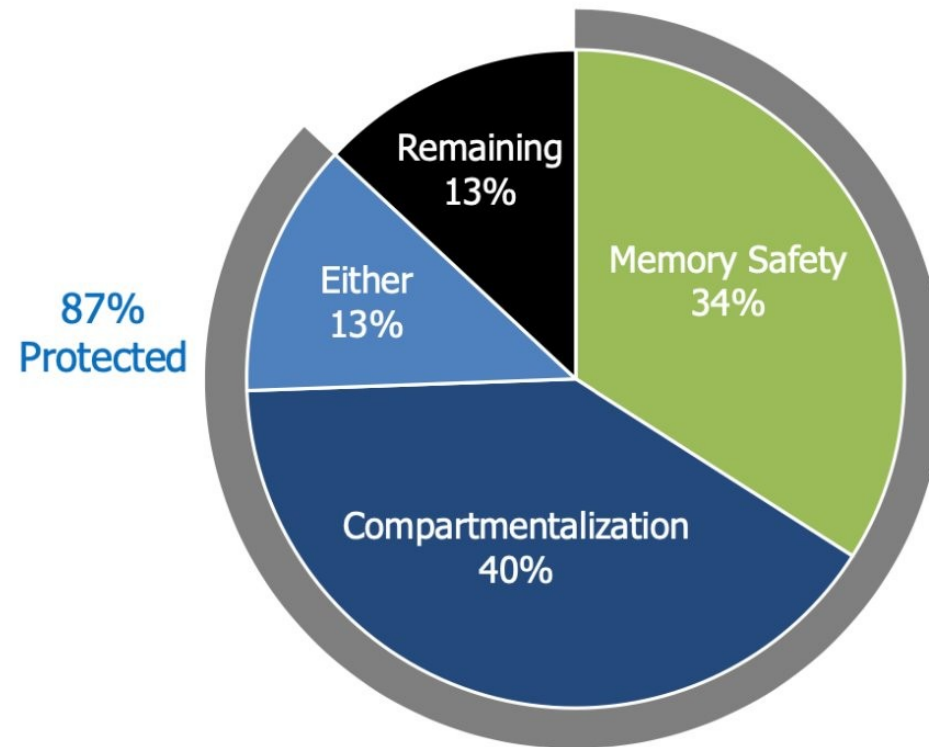


→ CHERI and Linux



Memory safety and compartmentalization would stop most campaigns

Mitigation analysis of Linux kernel high severity CVEs



- Either: if memory safety or compartmentalization were present, the bug would not be exploitable
- Remaining: memory safety or compartmentalization would not mitigate the CVE

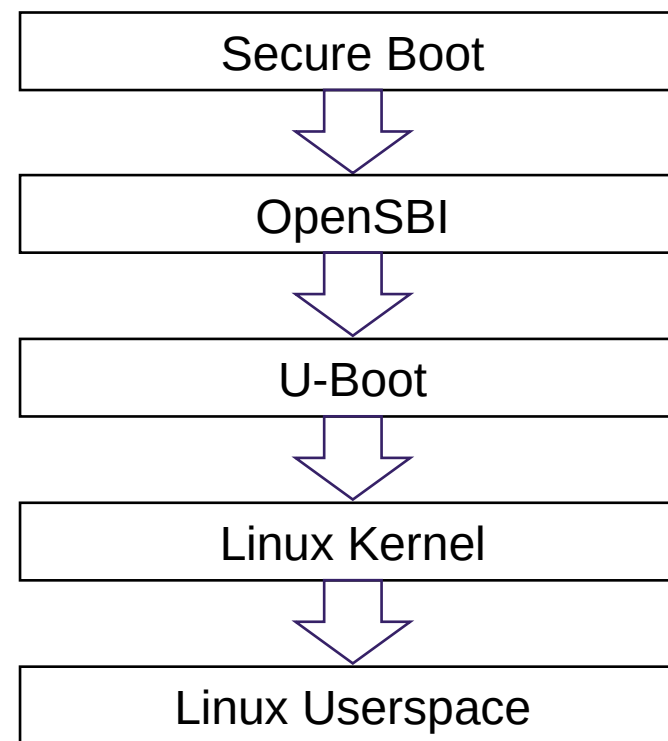
CVE: Common Vulnerabilities and Exposures

McKee, Derrick, et al. "Preventing kernel hacks with HAKC." Proceedings 2022 Network and Distributed System Security Symposium. NDSS. Vol. 22. 2022.

→ CHERI and Linux

So where are we currently?

- Pure capability v6.16 kernel
 - Used hybrid Morello Linux kernel as a starting point
 - Codasip team developing CHERI-RISC-V support
 - § Moving target as standardization is still on-going
 - Aiming for complete functionality
 - Testing against LTP
 - § Passes all syscall tests
- Userspace
 - Building up a pure capability userspace
 - No glibc yet, but musl
 - Concentrating on core libraries and utilities



Purecap embedded Linux boot flow

→ CHERI and Linux

Feature	Status
Module loading	Working - undergoing internal review
io_uring	Initial support - starting to pass tests
AIO	Working
Networking	Working (including wireless and wireless AP)
BPF/eBPF	Basic support
Tracing	Working on dynamic ftrace
Compat	32/64 bit, CHERI/non-CHERI
USB	Working (mouse, audio, keyboard, storage)
PCIe	Working (NVMe, graphics cards [DRM not ported])
Optimization	Optimized memcpy() and user copy functions added

GitHub - CHERI-Alliance/linux at codasip-cheri-riscv

→ CHERI and Linux

We need a new kernel/user ABI

- We use the PCuABI (pure capability userspace ABI) developed by Arm Morello over more than 3 years
- User to kernel:
 - userspace supplies capabilities through syscalls
 - `copy_to_user()/copy_from_user()` constrained to use those capabilities to transfer data
 - -EFAULT if the capability does not authorize the transfer
- Kernel to user:
 - Kernel returns capability to userspace (typically `mmap`) with limited bounds and permissions

§

→ CHERI and Linux

Challenges

- Linux moves rapidly : we use *clang-tidy* to automate migration
- CHERI-RISC-V specification is a moving target (but reaching maturity)
- Kernel uses *unsigned long* everywhere for pointers, we need to change these to *uintptr_t* pointer types
- Some structures assume pointers have a specific size or alignment, so we need to modify for the larger capabilities
- How to minimize changes to kernel APIs?
- How to modify code in a way we believe will be best for future upstreaming?

→ The future

→ The future

Next steps:

- Compartmentalized module loading
- eBPF
 - How to support eBPF compilation and bytecode to add capabilities?
 - How to support backwards compatibility?
- Userspace: compartmentalized user-space libraries

The work already done in CheriBSD is helpful

→ The future

MMU-less Linux

- Lack of process isolation has been one of the major blocking points to adoption
- CHERI can provide isolation instead of an MMU
- Still the issue of memory management

→ The future

Single address space Linux

- The MMU currently provides memory translation and isolation
- Use CHERI for isolation
- Beneficial when multiple independent processes need to share data

Configurable, 9-stage, dual-issue, in-order
CHERI-enabled application core.

*Compared to Cudasip A730, same core configured without CHERI and with 16 PMP regions
** Limited benchmarking so far



→ The future

Properties of Multiprogrammed Computer Systems

(5) An MCS must evolve to meet changing requirements.

An MCS does not exist in a static environment. Changing objectives, increased demand for use, added functions, improved algorithms and new technologies all call for flexible evolution of the system, both as a configuration of equipment and as a collection of programs.



Thank you!

carl.shaw@codasip.com

<https://codasip.com/solutions/riscv-processor-safety-security/cheri/x730-risc-v-application-proc>

<https://github.com/CHERI-Alliance>

<https://cheri-alliance.org>