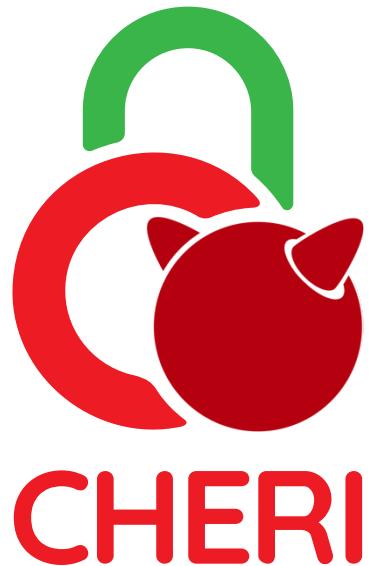


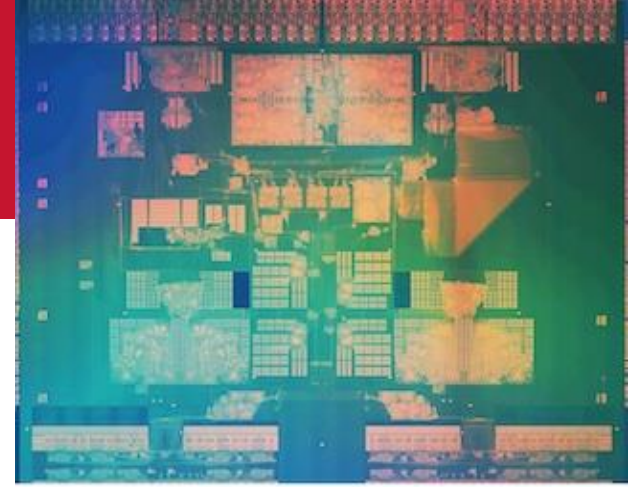
# Bringing memory safety to BSD with CHERI

BSDCan 2026, Ottawa, Canada



Brooks Davis  
Capabilities Limited

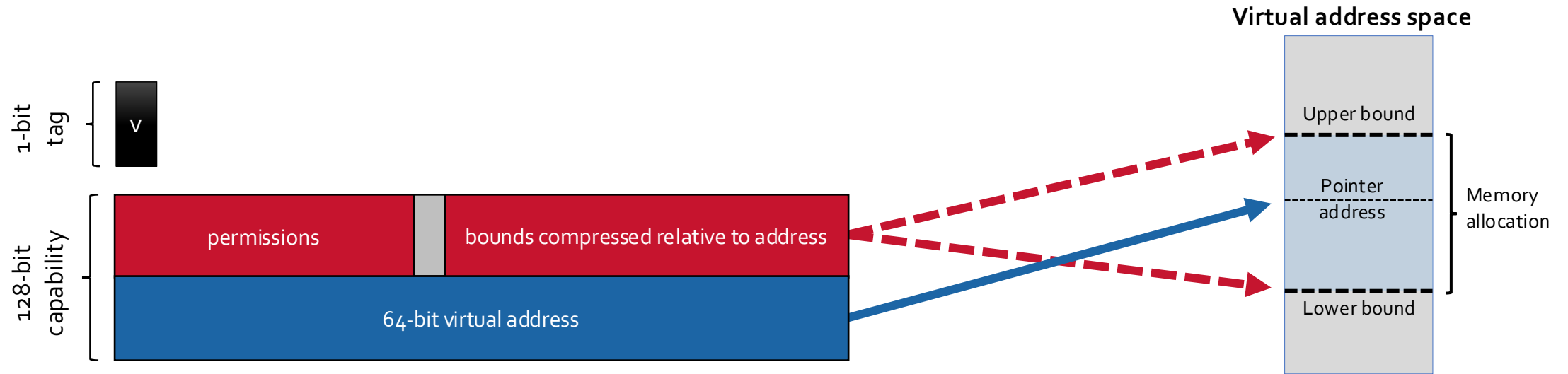
# What is CHERI



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

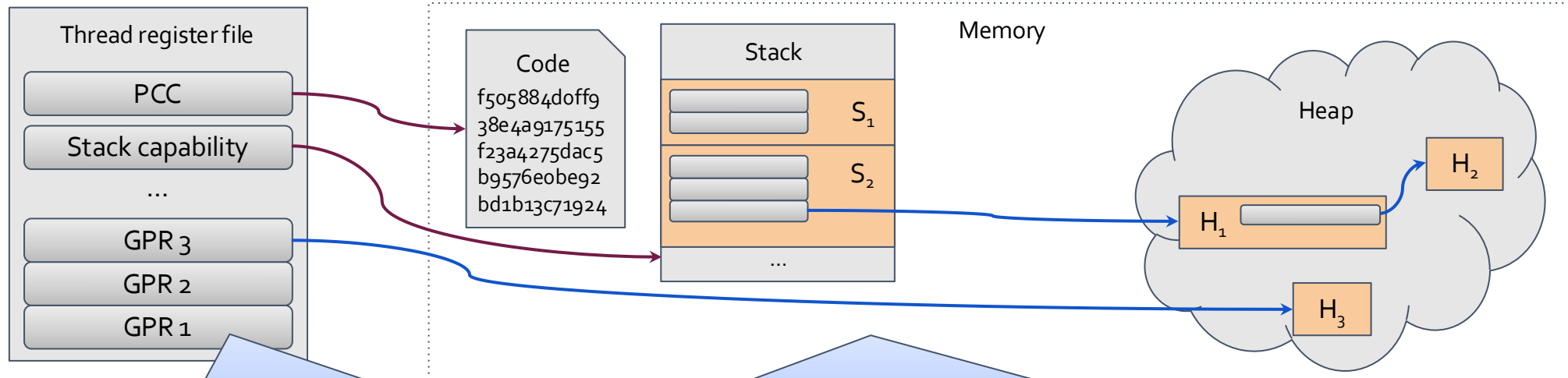
- CHERI is a processor **architectural protection model**
  - Composes a **capability-system model** with hardware and software
  - Adds new security primitives to Instruction-Set Architectures (ISAs)
  - Implemented by microarchitectural extensions to the CPU and SoC
  - Enables new security behavior in software
- CHERI mitigates vulnerabilities in **C/C++ Trusted Computing Bases**
  - Hypervisors, operating systems, 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** (Arm, Microsoft, Cudasip, SCI, SeqAI, ...)

# CHERI 128-bit capabilities



- **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
- Hardware guarantees correct usage

# Implementing C/C++ memory safety with CHERI (1/2)

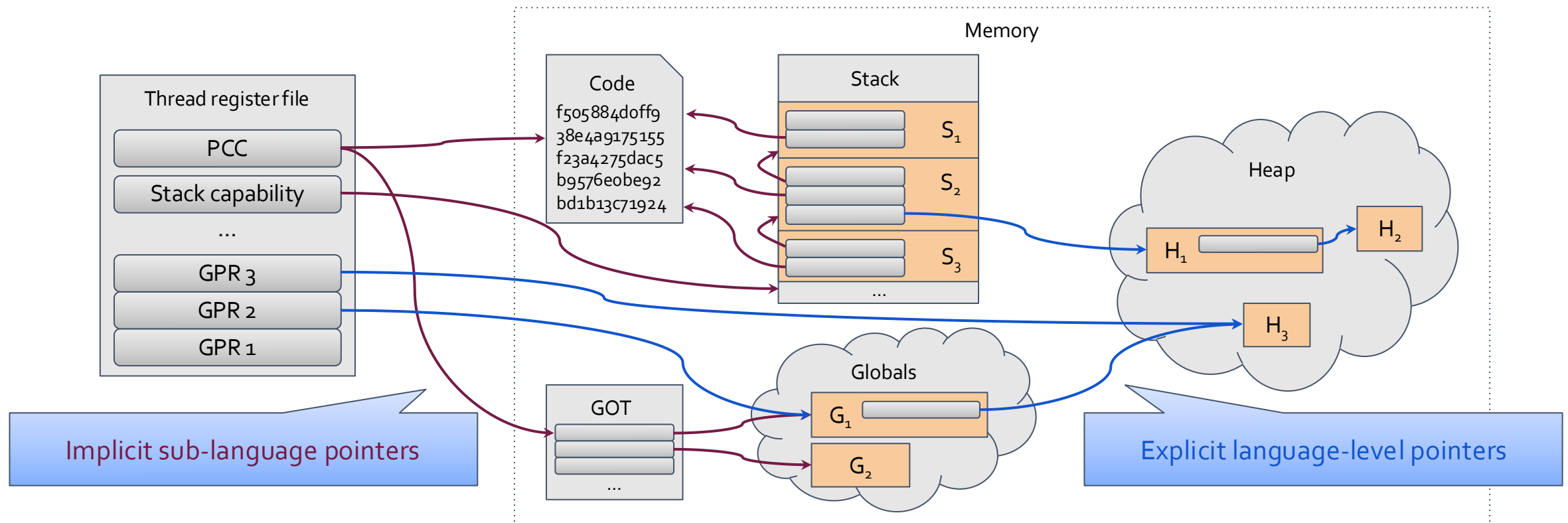


General-purposes registers and the program counter (PC) are extended to capability width and have tags.

Tagged memory preserves capability tag values flowing into and out of registers. Tags are implicitly cleared if non-capability stores overwrite tagged values.

- Whereas conventional C/C++ use **integer pointers**, CHERI C/C++ uses **capability pointers**
  - C/C++ data types are laid out to hold wider capability-width, capability-aligned pointers
  - Code generation is pointer-aware and uses explicit load/store-capability instructions
- Software TCBs restrict capability bounds and permissions as code runs
- Hardware continuously enforces capability protections on all pointer manipulation and use

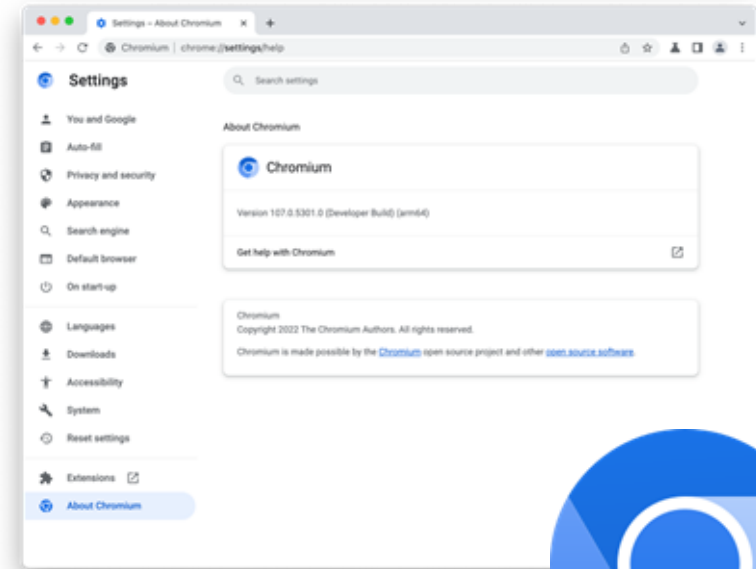
# Implementing C/C++ memory safety with CHERI (2/2)



- Protections are applied to all pointer types in compiled code and the run-time environment:
  - **Implicit sub-language pointers** such as GOT entries, stack pointers, return addresses
  - **Explicit language-level pointers** to globals, stack and heap allocations, functions
- Software TCBs refine **bounds** and **permissions** during execution
  - E.g., the mmap() system call, run-time linker, heap allocator, and stack allocator

# Grand challenge application: Google Chromium

- Foundation for Google Chrome, Microsoft Edge, Microsoft Teams, Electron, ...
  - ~**27MLoC** base source code with (at least) **7** compilers, of which **6** are Just-In-Time compilers (JITs)
  - ~**47MLoC** including >**760** library dependencies
  - V8, an intimidatingly real runtime for JavaScript and WASM (~**2MLoC**)
  - Code from diverse origins and in idiomatic C and C++
  - Vast wealth of past vulnerabilities to use in evaluation (~**1** critical memory-safety vulnerability every 2 days in 2023)
  - Performance critical components, especially V8
- Chromium able to browse increasingly complex web sites
  - Roughly 18 staff months of effort so far; close to functional browsing
  - .. but .. compare to size of **100-member Chromium security team!**
  - Chromium base (27MLoC) ~**0.045%** LoC change; V8 (2MLoC) ~**0.8%** LoC change
- Pilot project funded by UKRI, Google, DARPA



# Chromium: CVE-2023-4863 (“BLASTPASS”)



Aw, Snap!

This webpage detected a CHERI memory-safety protection violation and was terminated.

Error code: RESULT\_CODE\_GPU\_EXIT\_ON\_CONTEXT\_LOST

[Learn more](#)

Reload

**CHERI deterministically mitigates this Chromium vulnerability without any awareness about the nature, location, or origin of the vulnerability during development.**

This memory-safety vulnerability was discovered “in the wild” following targeted attacks against victims in the US using NSO Group’s Pegasus:

- **Naturally occurring vulnerability** in Google’s libwebp image library
  - Heap-memory buffer overflow exploitable for remote arbitrary code execution
  - Undiscovered for years despite fuzzing due to complexity of Huffman coding logic
- Affected Chrome, Edge, and WebKit
  - 1st-party code for Google
  - 3rd-party for Apple and Microsoft
  - Zero interaction exploitation of Apple iOS
- No prior awareness of this CVE in our work
- 0% LoC changes to webp for use on CHERI

# AlxCC nginx evaluation with CHERI

DARPA AI Cyber Challenge (AlxCC) applies AI to vulnerability discovery and patching

- Reintroduced past vulnerabilities / created new vulnerabilities, in open-source software – Challenge Problems (CPs)

One such problem adds vulnerabilities to the open-source nginx web server

- Already running with CHERI spatial/temporal safety following DSTL study
- AlxCC added 14 memory-safety Challenge Problem Vulnerabilities (CPVs)
- ASAN reproduction trigger provided for each CPV

The AlxCC team didn't know about CHERI, and the CHERI team didn't know about the AlxCC nginx work – so this is a great science experiment:

- **Query:** How would CHERI fare in mitigating vulnerabilities selected without regard to the protection approach?
- **Methodology:** Adversarial analysis (e.g., exploit aarch64 version, and attempt exploitation for CHERI version)

CPV					
1	heap-buffer-overflow	CVE-2009-2629	yes	yes	
2	heap-buffer-overflow	Synthetic	yes	yes	
3	heap-buffer-overflow	Synthetic	yes	yes	
4	heap-buffer-overflow	Synthetic	yes	yes	
5	SEGV	Synthetic	yes *	yes *	Unlikely to be directly exploitable for RCE on aarch64
8	heap-buffer-overflow	Synthetic	yes	yes	
9	heap-use-after-free	Synthetic	no	yes	Mitigation argument required code-specific analysis as revocation is deferred while quarantining
10	double-free	Synthetic	yes **	yes	
11	heap-use-after-free	Synthetic	no	yes	Mitigation argument required code-specific analysis as revocation is deferred while quarantining; Aliasing blocked but some information leak still
<del>12</del>	<del>heap-buffer-overflow</del>	<del>Synthetic</del>	<del>-</del>	<del>-</del>	<del>Not implemented on FreeBSD</del>
13	SEGV	Synthetic	yes *	yes *	Unlikely to be directly exploitable for RCE on aarch64
14	global-buffer-overflow	Synthetic	yes	yes	
15	SEGV	Synthetic	yes	yes	Sub-object bounds provide better protection
17	heap-use-after-free	Synthetic	no ***	yes	Mitigation argument required code-specific analysis

\* On Morello, page faults take priority over tag faults for NULL pointer dereferences, but both are checked architecturally

\*\* Double frees are safely disregarded in CheriBSD 24.05; from CheriBSD 25.03 onwards, they will trap immediately

\*\*\* Use-after-free is followed by a double free that does trap; see also \*\*

# Bonus discoveries

					Notes
CPV 15	-	Spatial safety vulnerability is a sub-object overflow	yes	yes	ASAN aborts because the provided trigger overflows the full heap allocation, but in fact we believe this vulnerability can be triggered by overflowing only the sub-object
BCPV 1	-	Linear buffer overflow	yes	yes	Occurs after a double free in CPV 10 in which ASAN had stopped execution, but also reachable via other inputs
BCPV 2	-	Null byte poisoning	yes	yes	One-byte overflow overwrites string terminator in AlxCC added feature

- Using reproduction triggers for two of the CPVs, we identified:
  - CPV15: A sub-object overflow could be exploited using input that would not have caused ASAN to fault
  - BCPV 1: An additional (and likely accidental) exploitable spatial-safety vulnerability (CPV 10)
  - BCPV 2: An additional (and likely accidental) one-byte overflow in an AlxCC added feature
- These shed light on the differences between ASAN and CHERI notions of memory safety and bug finding vs vulnerability mitigation, as well as the risks even expert developers experience when working with C/C++

# FreeBSD-SA-26:08.rpcsec\_gss (2026-03-26)

Forbes

INNOVATION > AI

## AI Just Hacked One Of The World's Most Secure Operating Systems

By [Amir Husain](#), Contributor. © Founder WQ Foundry, SkyGrid (acq. Boeing), ...

[Follow Author](#)

Published Apr 01, 2026, 08:38pm EDT, Updated Apr 01, 2026, 08:39pm EDT

Calif

### MAD Bugs: Claude Wrote a Full FreeBSD Remote Kernel RCE with Root Shell (CVE-2026-4747)

To our knowledge, this is the first remote kernel exploit both discovered and exploited by an AI.

“On CHERI, strong C/C++ memory safety deterministically traps on CHERI with a bounds violation when the attacker tries to trigger the stack buffer overflow **before** memory corruption can occur.”





<https://cheri-alliance.org/cheri-memory-safety-mitigates-llm-discovered-vulnerability-in-freebsd/>



# The latest FreeBSD security advisories (2026-06-09)

**Caution:** Early, low-effort analysis

CHERI mitigates a significant portion of security advisories:

-  spatial safety
-  overflow leading to spatial safety
-  Temporal safety (hypothetically)
-  Logic bugs

Fairly typical of past analysis



Security advisory	CHERI fixes?
FreeBSD-SA-26:25.thr	✗
FreeBSD-SA-26:26.ktls	✗
FreeBSD-SA-26:27.sound	✓ / ●
FreeBSD-SA-26:28.capsicum	✗
FreeBSD-SA-26:29.ip6_multicast	●
FreeBSD-SA-26:30.linux	✗
FreeBSD-SA-26:31.arm64	✗
FreeBSD-SA-26:32.elf	✓ (N/A)
FreeBSD-SA-26:33.unbound	✓ / ✗
FreeBSD-SA-26:34.vt	✓
FreeBSD-SA-26:35.openssl	✓ / ✗
FreeBSD-SA-26:36.ldns	✗

# CHERI and legacy applications side-by-side

← 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

The screenshot displays a desktop environment with several windows. On the left, a PDF viewer shows a document titled 'What is CHERI?' with a list of bullet points. In the center, a terminal window shows C code for a memory management example, with a message indicating a 'CHERI protection violation'. On the right, a web browser displays the University of Cambridge website, specifically the CTSRD page. At the bottom, a code editor shows assembly code. A large grey arrow at the top points left, indicating the direction of migration. Blue lines connect the text labels on the left to the corresponding windows in the screenshot.

Legacy 64-bit Arm Chromium browser

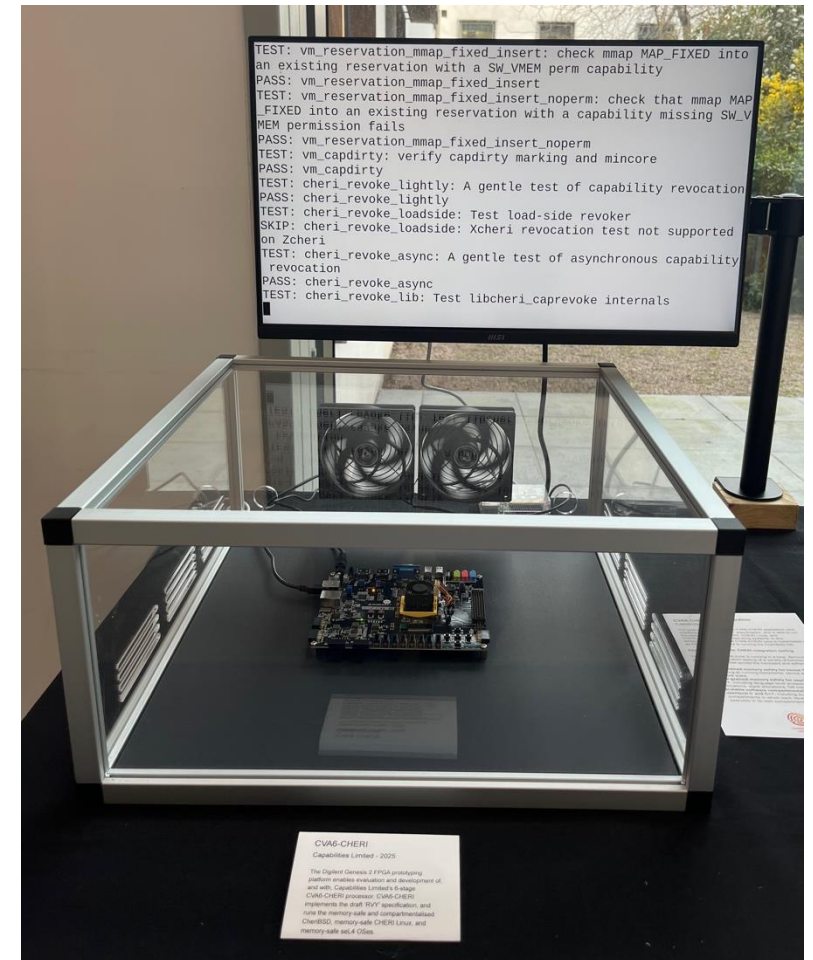
Legacy 64-bit Arm JVM

Memory-safe OS kernel and libraries



# CVA6-CHERI: Open-source RV64Y application core

- Base CVA6 is an open-source application core
  - OpenHW Foundation (Eclipse Group)
  - Under active development (incl. by Thales)
  - Production core written in SystemVerilog
  - 6-stage pipeline, dual-issue, MMU, multicore
- Extending CVA6 with CHERI RV64Y
  - Sufficient DV to allow CheriBSD to boot and run benchmarks on FPGA
  - Initial PPA improvements for CHERI
  - Multi-year R&D and upstreaming plan
- Initial prototyping project funded by the UK government
- New Innovate UK-funded joint COSMIC project with lowRISC and Oxford University to productionize as an open-source Secure Enclave product with formal verification



# Learning more about CHERI



- Visit [cheri.cst.cam.ac.uk](http://cheri.cst.cam.ac.uk)
- 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.

# Upstreaming CHERI support

Let's ship FreeBSD 16 with CHERI support!

Upstreaming CHERI support to FreeBSD is supported by Innovate UK and the Department for Science, Innovation and Technology for the adoption and diffusion of CHERI technology under project 10168042 ("CheriBSD feature extraction, maturity, and testing")

# Why upstream?

- FreeBSD needs a memory safety story
  - Virtually all OSes lack memory safety today
  - ... but that won't remain acceptable forever (government regulations, etc)
  - Rewriting it all isn't a practical answer (besides, many of us enjoy C)
- Reduce adoption risk (vs integrating a fork)
- Provide a production OS with proven features (vs a research OS)
- Changes generally improve code quality
  - Many clarify programmer intent
  - Testing with CHERI finds general bugs, some exploitable
- CheriBSD gives BSD a lead on CHERI adoption - Let's keep it!

# What we're upstreaming

- RV64Y (risc64c) – RISC-V CHERI standard (Tier 2)
  - Hardware on the way, FPGA available
- Arm Morello – Prototype armv8.2 + CHERI (Tier 3)
  - Prototype hardware exists, allows direct development, good demos
- Focus on pure-capability CHERI C/C++ programming
  - Most secure and least disruptive to the source tree
  - Sub-object bounds to follow
- Kernel support leads userspace

# Timeline of patch production

- September 2025 – March 2026: pure-capability kernel support\*
  - Goals: spatially safe kernel supporting FreeBSD and CheriBSD userspace
- April 2026 – September 2026: pure-capability userspace support\*
  - Spatially safe userspace on FreeBSD/CHERI kernel
- October 2026 – February 2027: linker-based compartmentalization†
  - Enables stronger, finer grained isolation of components
- March 2027 – June 2027: Heap temporal safety†
  - Eliminates use-after-reallocation temporal safety issues

\* Critical functionality with low adoption risk

† More disruptive changes with higher adoption risk

# Progress to date

```
root@freebsd-aarch64:~ # uname -a
FreeBSD freebsd-aarch64 16.0-CURRENT FreeBSD 16.0-CURRENT #2 ca-cheri-n286377-be92f0f6ab75: Sat
Jun 20 15:29:32 BST 2026      xxxxx@xxxxx.cl.cam.ac.uk:/home/xxxxx/cheri/build/freebsd-aarch64-
build/home/xxxxx/cheri/ca-cheri/arm64.aarch64/sys/GENERIC-MORELLO arm64
root@freebsd-aarch64:~ # sysctl kern.supported_archs
kern.supported_archs: aarch64c aarch64
```

- Booting to multi-user on Morello and RVY (vo.9.3) with unmodified FreeBSD userspace (aarch64)
  - Pure-capability binaries (from CheriBSD) run and most tests pass
  - Userspace in progress (libraries largely merged, now working on rtld)
- ~450 commits in current WIP patch
- 746 files changed, 54895 insertions(+), 5778 deletions(-)
  - Generated portion: 15 files changed, 16276 insertions(+), 58 deletions(-)

# Tidbit: CHERI and programmer intent

- Capability systems such as CHERI implement two key 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
- New APIs to make it clear when we do or don't propagate capabilities
  - `copy{in,out}` does not. `copy{in,out}ptr` does.
  - `memcpy` does. `memcpy_data` does not.
- Review stack (5 reviews):
  - `memcpy_data`: <https://reviews.freebsd.org/D57662>
  - `copyinptr, ...`: <https://reviews.freebsd.org/D57663>

# Tidbit: freebsd64 (legacy 64-bit compatibility)

- Inspired by freebsd32 (legacy 32-bit compatibility)
  - ... but different
- Under freebsd32, lot of sizes change
  - Pointers
  - `long` → (`size_t`, `uintptr_t`, `long`)
  - Sometimes `time_t` (x86 only)
  - Special handling for `off_t` (split across registers, maybe aligned)
  - See also: “So you want to add a syscall?”
- Under freebsd64, pointers change size and nature
  - Other types remain the same
- New wrappers needed to pass capabilities to implementations
- <https://github.com/freebsd/freebsd-src/pull/2284>

# Tidbit: freebsd64 (example wrapper)

```
int
sys_exterrctl(struct thread *td, struct exterrctl_args *uap)
+{
+    return (kern_exterrctl(td, uap->op, uap->flags, uap->ptr));
+}
+
+int
+kern_exterrctl(struct thread *td, u_int op, u_int flags, void *ptr)
{
    uint32_t ver;
    int error;

-    if ((uap->flags & ~(EXTERRCTLF_FORCE)) != 0)
+    if ((flags & ~(EXTERRCTLF_FORCE)) != 0)
        return (EINVAL);

```

# Tidbit: freebsd64 (example wrapper call)

```
int
freebsd64_exterrctl(struct thread *td, struct freebsd64_exterrctl_args *uap)
{
    return (kern_exterrctl(td, uap->op, uap->flags,
        USER_PTR(uap->ptr, sizeof(struct uexterror))));
}
```

- Same as `sys_exterrctl`, except we create a (possibly bounded) capability with the `USER_PTR` macro

# Tidbit: networking ioctls (struct ifreq)

- ioctl commands are declared as a group, number, and type (encoded as size):

```
#define SIOCGIFMAC _IOWR('i', 38, struct ifreq)
```

- Usually, if the type contains pointers or other things requiring adaptation, the size changes
  - `struct ifreq` contains pointers in a union, but the size does not change between 32-bit and 64-bit (the union is 16-bytes!)
    - Special handling still required so we use process ABI (SV\_ILP32)
  - On CHERI, the size does change!
    - There are lots of ioctls using `struct ifreq`
    - Helper macro: `CASE_IOC_IFREQ`
      - `"case SIOCGIFMAC:"` becomes `"case CASE_IOC_IFREQ(SIOCGIFMAC):"`
    - Accessors like `ifr_data_get_ptr()`

# git strategy

- Fork of the freebsd-src repo under the CHERI Alliance github
  - <https://github.com/cheri-alliance/freebsd-src>
- Maintain **rebased** branch(es) of upstreaming candidates
  - `main` tracks upstream FreeBSD
  - `main-ci` adds CI from The Capable Hub (currently broken)
  - `cheri` contains patches adding CHERI
    - *Warning: patch readiness varies!*
- Patches or patch groups to be submitted via Phabricator or GitHub pull requests as appropriate
- Some patches arriving directly
  - (void \*) pointers for vm APIs (merged)  
<https://github.com/freebsd/freebsd-src/pull/2068>

# Please get involved!

- We aim to make FreeBSD/CHERI the reference POSIX-like OS for CHERI
  - We're broadly confident in our approach, but more eyes will help make it better
- Most reviews in phabricator (reviews.freebsd.org) tagged with `#cheri`
  - Commits tagged with "Effort: CHERI upstreaming"
- Occasional pull requests for example <https://github.com/freebsd/freebsd-src/pull/2284>
- Run a different BSD? Consider adding CHERI support!

# Q&A