# CHERI projects overview

Alex Richardson

This document is a brief collection of various CHERI-related information, e.g. Documentation, hardware implementations and emulators, the available CHERI-enabled software stack (and the underlying toolchain infrastructure), supported programming languages, and a short list of new security (and performance) features that are possible using the CHERI architectural primitives.

If something is missing/not clear or if you would like to know whether a given open-source project has been ported to CHERI please leave a comment or message Alex Richardson.

# Documentation

## CHERI architecture documentation

An Introduction to CHERI

This report is a fairly high-level overview of CHERI. It was published in 2019 and therefore does not mention Morello, but is still a good starting point.

CHERI Instruction-Set Architecture (Version 8)

This is the latest version of the detailed reference manual (almost 600 pages). Describes the architecture-independent aspects of the CHERI protection model, the CHERI RISC-V (and MIPS) implementations, microarchitectural techniques, instruction references, various experimental features/design ideas.

Adversarial CHERI Exercises and Missions

A few exercises to explore CHERI memory protection features. Also includes some more complex exercises for uninitialized memory and temporal safety.

## Arm's Morello documentation

Arm Architecture Reference Manual Supplement - Morello for A-profile Architecture

Overview of the Morello capability format, instruction opcodes+semantics, etc.

https://git.morello-project.org/morello/docs

Various documents describing how to get started with Android and baremetal Morello development.

https://www.cl.cam.ac.uk/research/security/ctsrd/cheri/cheri-morello.html

Various Morello resources from Cambridge.

## Porting C/C++ programs to CHERI

CHERI C/C++ Programming Guide

This report gives an overview over the things to look out for in CHERI C/C++. It also documents various C/C++ intrinsics and the sub-object bounds feature.

Assessing the Viability of an OpenSource CHERI Desktop Software Ecosystem

This recent report (Aug 2021) includes detailed analysis of changes required to run a KDE Plasma desktop on top of CheriBSD on CHERI-RISC-V and Morello - very few no changes were required (0.0026% SLoC modified)

CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment

This paper describes the changes needed in an operating system kernel+userspace to support CHERI capabilities (e.g. context switching/system calls/swap/etc.)
**Important:** the statistics (and some descriptions) of the required changes are partially out of date because compiler improvements have removed the need for many of them.

# Hardware implementations and emulators

There are multiple emulators and FPGA implementations that can be used to run CHERI code. In general, QEMU is the easiest and fastest tool, but depending on use-cases the other implementations are also useful.

## CHERI-QEMU

- Fork of QEMU maintained by the University of Cambridge that adds CHERI support
- Supports 32- and 64-bit CHERI-RISC-V, Morello (currently only in the dev branch), and CHERI-MIPS (deprecated, may be removed eventually)
- Fastest available emulator for all CHERI-enabled architectures
- No binary releases (a somewhat supported docker image exists), building it locally with cheribuild is recommended
- Source code: https://github.com/CTSRD-CHERI/qemu

## Arm Morello FVP

- Official Morello emulator
- Significantly slower than QEMU but more realistic whole-system emulation (for example, it also emulates the SCP and MCP and can be used to develop that firmware).
- Like QEMU the FVP is not cycle-accurate
- Can be downloaded for x86 Linux here
- Alternatively, cheribuild provides infrastructure to install and run it on macOS using docker

## CHERI-RISC-V FPGA implementations

The University of Cambridge maintains three RISC-V implementations:

- Tooba
  - Out-of-order 64-bit CHERI-enabled RISC-V core
  - Bluespec SystemVerilog source code: https://github.com/CTSRD-CHERI/Toooba
- Flute
  - 64-bit CHERI-enabled RISC-V core with in-order 5-stage pipeline
  - Bluespec SystemVerilog source code: https://github.com/CTSRD-CHERI/Flute
- Piccolo
  - 32-bit CHERI-enabled RISC-V core with in-order 3-stage pipeline
  - Bluespec SystemVerilog source code: https://github.com/CTSRD-CHERI/Piccolo
  - **Barely any active development** - bug fixes and updates to shared code only

## Sail formal model for CHERI-RISC-V

- Formal model written in the Sail architecture definition language

- Generates 32- and 64-bit CHERI-RISC-V simulators from the ISA description (very slow compared to QEMU)
- Also used to generate parts of the CHERI ISA documentation and theorem prover definitions
- Source code: https://github.com/CTSRD-CHERI/sail-cheri-riscv

## Sail formal model for Morello

- Similar to the RISC-V version but auto-generated from Arm's architecture specification with only a few hand-written changes
- Not (yet?) available as open-source

# Software toolchain

## cheribuild

- Script to automate building and running most of the CHERI and (non-Android) Morello software stack
- Can be used to build e.g. the toolchain+QEMU emulator+CheriBSD operating system as well as lots of 3rd-party applications that have been ported to CHERI C/C++ (including e.g. WebKit or the KDE Plasma desktop plus dependencies)

## CHERI LLVM/Clang/LLD

- Compiler and linker for CHERI-RISC-V (and deprecated CHERI-MIPS)
- Supports both pure-capability and hybrid compilation modes for C and C++
- Currently based on LLVM 13
- Source code: https://github.com/CTSRD-CHERI/llvm-project

## Morello LLVM/Clang/LLD/LLDB

- Arm's fork of CHERI LLVM that adds support for Morello
- Unlike CHERI LLVM, the Morello version also has a port of the LLDB debugger for Linux
- Generally lags quite far behind CHERI LLVM - currently about 1 year of upstream changes behind CHERI LLVM
- (Currently) breaks RISC-V support, so should only be used for Morello
- Source code: https://git.morello-project.org/morello/llvm-project
- Linux x86 binaries: https://git.morello-project.org/morello/llvm-project-releases
  Note: these are **usually rather outdated**, so building from source is recommended

## Morello GCC

- Arm also has a team working on Morello support for GCC
- Extremely experimental, not sure if recent source code is available yet
- Pure-capability compilation only

## CHERI GDB

- GDB with CHERI support for CHERI-RISC-V (and partially Morello) maintained by the University of Cambridge
- Can be used to debug CheriBSD binaries or baremetal applications
- No binutils support (CHERI-RISC-V is LLVM only)
- Source code: https://github.com/CTSRD-CHERI/gdb

## Morello GDB and GNU binutils

- Arm maintains a version of GDB with support for debugging Morello Linux binaries
- Unlike CHERI GDB, this includes binutils supports (assembler works, not sure if ld.bfd is supported)
- Source code: https://git.morello-project.org/morello/binutils-gdb

## Morello firmware

- Arm provides version of their firmware for Morello boards (also needed for the FVP, not needed for QEMU)
- IIRC these are minimal adaptations rather than making extensive use of CHERI
- Source code:
    - https://git.morello-project.org/morello/scp-firmware
    - https://git.morello-project.org/morello/trusted-firmware-a
    - https://git.morello-project.org/morello/edk2
    - https://git.morello-project.org/morello/edk2-platforms

# CHERI-enabled operating systems/software stacks

## CheriBSD (FreeBSD with CHERI support)

- CheriBSD is the most mature software platform for CHERI development
- Supports CHERI-RISC-V and Morello (successfully boots on the real Morello hardware)
- Fully working FreeBSD base system with all common UNIX utilities
- Lots of third-party software can be run on top of this platform (including X11 GUI - albeit currently only over VNC)
- Maintained by the University of Cambridge and SRI
- Easiest way to run it is with cheribuild
- Source code: https://github.com/CTSRD-CHERI/cheribsd
- Binary release snapshot: https://cheri-dist.cl.cam.ac.uk/releases/2021.08/relnotes.html (requires docker to run)

## Linux kernel

- Linux kernel with basic support for Morello (e.g. context switching of capability registers)
- System call ABI in progress, but currently relies on Morello libshim for CHERI system calls
- Currently based on Android 12 kernel (Linux 5.10)
- Source code: https://git.morello-project.org/morello/kernel/morello-ack/
- Implementation questions best directed at Kevin.Brodsky@arm.com

## Android

- Currently based on Android 11 (update to 12 in-progress)
- Available from https://git.morello-project.org/morello/android (e.g. Bionic)
- I have never tried running it so very limited knowledge on the degree of CHERI-adaption, for further details I'd recommend asking Ruben.Ayrapetyan@arm.com
- More information here: https://git.morello-project.org/morello/docs/-/blob/morello/mainline/android-readme.rst

## CheriOS

- Single-address space microkernel OS written from scratch for CHERI-enabled architectures (currently only for MIPS, but ongoing work to port to RISC-V & Morello)
- Highly compartmentalized system
- Complete distrust between components, only TCB is the "nanokernel"
- For design choices, etc. see Lawrence Esswood's PhD dissertation: CheriOS: designing an untrusted single-address-space capability operating system utilising capability hardware and a minimal hypervisor
- Source code: https://github.com/CTSRD-CHERI/cherios

## FreeRTOS

- A version of FreeRTOS with support for compartmentalization
- **Experimental status -** will be described in Hesham Almatary's upcoming PhD thesis
- Source code: https://github.com/CTSRD-CHERI/FreeRTOS/tree/hmka2
- Needs a custom toolchain (https://github.com/CTSRD-CHERI/llvm-project/tree/cherifreertos-gprel-dev) and separate branch of cheribuild to compile

## RTEMS

- Port of the RTEMS Realtime SMP Kernel
- This is a basic port to CHERI-RISC-V by Hesham Almatary - **not currently maintained**
- Source code: https://github.com/CTSRD-CHERI/rtems

## Baremetal Morello newlib

- Port of newlib to allow running pure-capability bare-metal applications
- Source code: https://git.morello-project.org/morello/newlib

# Programming languages and language runtimes

## C and C++

- Fully supported using the CHERI or Morello LLVM compilers

## Objective-C/C++

- **Unmaintained** - Clang/LLVM should still work, but the https://github.com/CTSRD-CHERI/libobjc2 runtime was last update 2017 and only ever ported to MIPS

## JavaScript runtimes

### WebKit

- Various JavaScript test suites and benchmarks work correctly
- Includes basic JIT support for Morello - interpreter-only for other architectures
- Source code: https://github.com/CTSRD-CHERI/webkit
- Older QtWebKit port available at https://github.com/CTSRD-CHERI/qtwebkit

### QtQML

- Qt's QML declarative UI language includes a JavaScript runtime
- Port of version 5.15 can successfully run almost all tests on CheriBSD and is used extensively when running a KDE Plasma desktop
- Source code: https://github.com/CTSRD-CHERI/qtdeclarative

## Python

- Basic port of Python 3.8
- **Proof-of-concept level** - stopped working on it once Hello World ran
- Source code: https://github.com/CTSRD-CHERI/cpython

## Perl

- Basic port of Perl 5.32
- Maturity level unclear - for more information contact Jessica.Clarke@cl.cam.ac.uk
- Source code: https://github.com/CTSRD-CHERI/perl5

## Bash

- Can be used as an interactive shell on CheriBSD, but quite lightly tested

- Source code: https://github.com/CTSRD-CHERI/bash

## Java

- University of Manchester is working on porting OpenJDK to CHERI
- **Very early prototype stage - not working yet**

# Miscellaneous CHERI utilities/libraries/etc.

## CHERI compiler explorer

- https://cheri-compiler-explorer.cl.cam.ac.uk/ is a version of godbolt.org that allows you to write C/C++/LLVM IR code and see the assembly code that LLVM generates for RISC-V and Morello in different compilation modes

## Morello encoding visualizer

- Visual decoding of a raw bitstring at https://www.morello-project.org/capinfo
- Source code: https://git.morello-project.org/morello/utilities/capinfo

## cheri-compressed-cap

- C library to decode/encode/manipulate in-memory representations of Morello and RISC-V (both 32 and 64 bit variants) capabilities
- Used internally in QEMU/GDB/LLVM
- Source code: https://github.com/CTSRD-CHERI/cheri-compressed-cap

## CHERI CAP LIB

- Bluespec SystemVerilog library for RISC-V CHERI capability formats - this is used by the CHERI FPGA implementations
- Partially description of the approach in the 2017 paper Efficient Tagged Memory
- Source code: https://github.com/CTSRD-CHERI/cheri-cap-lib

## TagController:

- Bluespec SystemVerilog library to support CHERI tagged memory using a reserved region of DRAM - used by the CHERI FPGA implementations
- Source code: https://github.com/CTSRD-CHERI/TagController

## Morello libshim

- Library that provides (insecure) CHERI versions of Linux systems calls to allow running software on top of a Linux kernel with very limited CHERI support
- Will be obsolete once CHERI system call support is added to Linux
- Source code: https://git.morello-project.org/morello/android/platform/external/libshim/

# Experimental CHERI-based security features

## (Heap) temporal memory safety

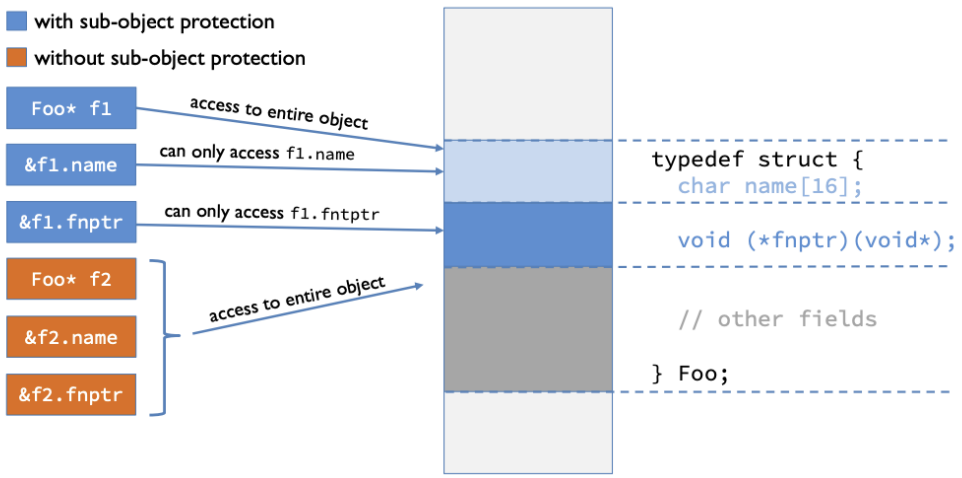
Compiling code for CHERI provides spatial memory safety out-of-the-box. In order to get temporal memory safety (specifically preventing use-after-reallocation) you currently have to use experimental operating system kernel support is needed. For CheriBSD, this code lives in the caprevoke branch of the CheriBSD repository.

The code supports two temporal safety mechanisms. Firstly there is the one described in the CHERIvoke and Cornucopia papers (the latter being older and simulation only). Secondly, Wes Filardo at Microsoft research is currently working on a (hopefully much) lower overhead approach. This approach uses a new architectural MMU feature (capability load generations) that is present in both Morello and RISC-V. It is documented in a report that is not yet publicly available. If there is interest in reading it, I can ask for permission to share the current draft.

Both approaches look at heap memory only, for stack memory other techniques need to be used. There is ongoing research into stack temporal safety at Cambridge, but the prototypes have so far either not been performant enough or required too many bits in the capability format.

## Sub-object bounds

The current default in Clang for C and C++ is to restrict capability bounds to the allocation level (e.g. the range returned by malloc() or the size of a stack-allocated object). While spatial safety prevents overflowing the structure and pointer injection is impossible, a possible CHERI-aware exploit could still overflow adjacent non-pointer data fields within a structure (for example set a user ID field to root).



Sub-object bounds can be enabled by passing a compiler flag Overheads are very low (just a single added instruction). The reason this is not enabled by default is that it does impact C/C++ compatibility in some cases (e.g. container_of() usage, etc.). For details on implementation and compatibility, see chapter 5 of Alex Richardson's PhD dissertation.

## Co-processes - multiple processes in the same address space

The experimental co-process feature allows multiple UNIX processes to share the same virtual address space (and therefore the same page tables and TLB entries) by relying on CHERI capability bounds for memory isolation instead of using the MMU. Once mature, it should be possible to use this to significantly reduce the overheads for certain compartmentalization models.

The source code for this can be found in the coexeve branch of CheriBSD and an IPC mechanism for co-processes is in the cocall branch. I don't think there is much documentation, while I can probably answer some high-level questions, for anything more detailed I'd recommend contacting trasz@freebsd.org or jhb@freebsd.org.