DSbD All Hands November 2023
Tutorial: New features in CheriBSD 23.11

Robert N. M. Watson, Simon W. Moore, Peter Sewell, Peter G. Neumann, Brooks Davis
Hesham Almatary, Ricardo de Oliveira Almeida, Jonathan Anderson, Alasdair Armstrong, Rosie Baish, Peter Blandford-Baker, John Baldwin, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, Brian Campbell, David Chisnall, **Jessica Clarke**, Nirav Dave, Lawrence Esswood, Nathaniel W. Filardo, Franz Fuchs, **Dapeng Gao**, Ivan Gomes-Ribeiro, Khilan Gudka, Brett Gutstein, Angus Hammond, Graeme Jenkinson, Alexandre Joannou, Mark Johnston, Robert Kovacsics, Ben Laurie, A. Theo Markettos, J. Edward Maste, **Alfredo Mazzinghi**, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, George Neville-Neil, Kyndylan Nienhuis, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Allison Randal, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Thomas Sewell, Stacey Son, Ian Stark, Domagoj Stolfa, Andrew Turner, Munraj Vadera, **Konrad Witaszczyk**, Jonathan Woodruff, Hongyan Xia, Vadim Zaliva, and Bjoern A. Zeeb

University of Cambridge and SRI International
Manchester, November 8, 2023

Approved for public release; distribution is unlimited. This research is sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contracts FA8750-10-C-0237 ("CTSRD"), with additional support from FA8750-11-C-0249 ("MRC2"), HR0011-18-C-0016 ("ECATS"), FA8650-18-C-7809 ("CIFV"), and HR001122C0110 ("ETC"). The views, opinions, and/or findings contained in this article/presentation are those of the author(s)/presenter(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

UNIVERSITY OF CAMBRIDGE

Agenda

Start: 2:00 PM	
Part I: presentation **(~25 min)**	Overview
	Benchmark ABI
	Heap temporal memory safety
	Library compartmentalization
	Visualization tooling
Part II: practice **(~35 min)**	Work environment setup (FreeBSD jails on CheriBSD/Morello hosts)
	Exercises
End: 3:00 PM	
Round 2 workshop: Measuring, analysing, and understanding performance on Morello 3:15 PM - 4:15 PM in International Suite (Conference end)	

Part I: presentation

New features in CheriBSD 23.11.

CHERI prototype software stack on Morello

- **Complete open-source software stack** from bare metal up: compilers, toolchain, debuggers, hypervisor, OS, applications – all demonstrating CHERI
- Rich CHERI feature use, but fundamentally incremental/hybridized deployment

Open-source application suite (KDE Plasma, Wayland, WebKit, Python, OpenSSH, nginx, …)

CheriBSD/Morello (funded by DARPA and UKRI)
(Morello and CHERI-RISC-V)

- FreeBSD kernel + userspace, application stack
- Kernel spatial and referential memory protection
- Userspace spatial, referential, and temporal memory protection
- Co-process compartmentalization (development branch)
- Linker-based compartmentalization
- Morello-enabled bhyve Type-2 hypervisor
- ARMv8-A 64-bit binary compatibility for legacy binaries

Android (Arm)
(Morello only)

Linux (Arm)
(Morello only)

CHERI Clang/LLVM compiler suite, Morello GCC, LLD, LLDB, GDB

Baseline CHERI Clang/LLVM from SRI/Cambridge; Morello adaptation by Arm + Linaro

Maturing CHERI software artifacts

Feature	Status	Availability
3rd-party packages (Hybrid)	23K **memory-unsafe** software packages with strong functionality expectations	Since May 2022 (22.05 release)
3rd-party packages (CheriABI)	11K **memory-safe** software packages with mixed functionality expectations	Since May 2022 (22.05 release) **Up from 9k packages in 23.11**
Morello GPU device drivers	Memory-safe kernel and user drivers,	Since December 2022 (22.12 release)
Benchmark ABI support (+3rd-party packages)	Support for modified code generation addressing Morello bounds prediction	**Shipping in 23.11** (roughly the same packages as CheriABI)
Userlevel heap temporal safety	Prototype implements strong temporal safety, developed with Microsoft; testing required	**Shipping in 23.11** (pretty experimental)
Linker-based compartmentalization	Introduces strong encapsulation boundaries around UNIX libraries with no modification	Since 22.12 (very experimental); **Significant improvements in 23.11**
bhyve (Type-2) hypervisor	Prototype boots pure-capability guest OS, validation required	**Shipping in 23.11** (very experimental)
Co-process compartmentalization	Prototype runs some compartmentalized software (e.g., OpenSSL); API co-design	Planning to ship in 2024

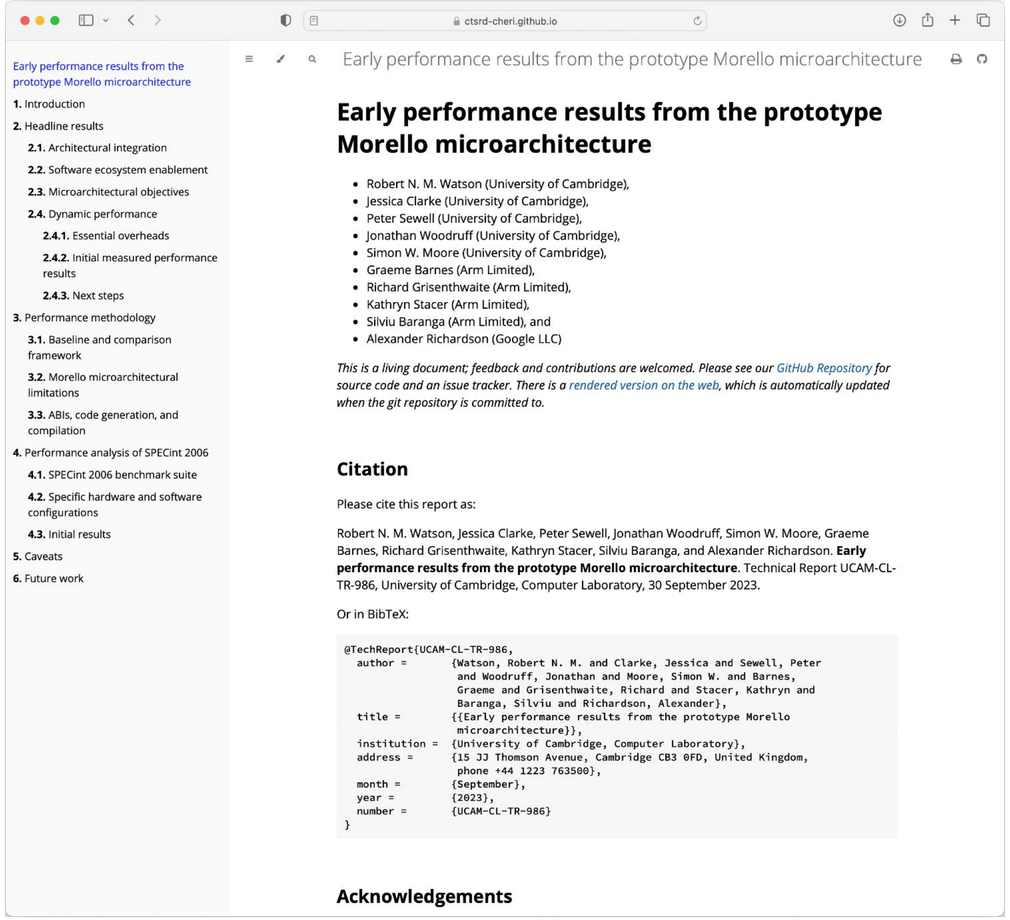
CheriBSD 23.11

- The release is expected to be published in the next two weeks – final integration and testing are in progress.

- The release and this workshop are focused on CheriBSD on Arm Morello, but the release is continuously tested not to break existing functionality on CHERI-RISC-V.

- NB: This tutorial is running the 2023-11-01 ‘dev’ snapshot, with a patch to the c18n run-time linker.

Early Performance Results from the Prototype Morello Microarchitecture



ctsrd-cheri.github.io

Early performance results from the prototype Morello microarchitecture

Early performance results from the prototype Morello microarchitecture
1. Introduction
2. Headline results
2.1. Architectural integration
2.2. Software ecosystem enablement
2.3. Microarchitectural objectives
2.4. Dynamic performance
2.4.1. Essential overheads
2.4.2. Initial measured performance results
2.4.3. Next steps
3. Performance methodology
3.1. Baseline and comparison framework
3.2. Morello microarchitectural limitations
3.3. ABIs, code generation, and compilation
4. Performance analysis of SPECint 2006
4.1. SPECint 2006 benchmark suite
4.2. Specific hardware and software configurations
4.3. Initial results
5. Caveats
6. Future work

Early performance results from the prototype Morello microarchitecture

- Robert N. M. Watson (University of Cambridge),
- Jessica Clarke (University of Cambridge),
- Peter Sewell (University of Cambridge),
- Jonathan Woodruff (University of Cambridge),
- Simon W. Moore (University of Cambridge),
- Graeme Barnes (Arm Limited),
- Richard Grisenthwaite (Arm Limited),
- Kathryn Stacer (Arm Limited),
- Silviu Baranga (Arm Limited), and
- Alexander Richardson (Google LLC)

This is a living document; feedback and contributions are welcomed. Please see our GitHub Repository for source code and an issue tracker. There is a rendered version on the web, which is automatically updated when the git repository is committed to.

Citation

Please cite this report as:

Robert N. M. Watson, Jessica Clarke, Peter Sewell, Jonathan Woodruff, Simon W. Moore, Graeme Barnes, Richard Grisenthwaite, Kathryn Stacer, Silviu Baranga, and Alexander Richardson. **Early performance results from the prototype Morello microarchitecture**. Technical Report UCAM-CL-TR-986, University of Cambridge, Computer Laboratory, 30 September 2023.

Or in BibTeX:

```
@TechReport{UCAM-CL-TR-986,
  author =       {Watson, Robert N. M. and Clarke, Jessica and Sewell, Peter
                  and Woodruff, Jonathan and Moore, Simon W. and Barnes,
                  Graeme and Grisenthwaite, Richard and Stacer, Kathryn and
                  Baranga, Silviu and Richardson, Alexander},
  title =        {{Early performance results from the prototype Morello
                  microarchitecture}},
  institution =  {University of Cambridge, Computer Laboratory},
  address =      {15 JJ Thomson Avenue, Cambridge CB3 0FD, United Kingdom,
                  phone +44 1223 763500},
  month =        {September},
  year =         {2023},
  number =       {UCAM-CL-TR-986}
}
```

Acknowledgements

https://ctsrd-cheri.github.io/morello-early-performance-results/

- Presents first detailed performance analysis on the first-generation Morello prototype
- Identifies and mitigates/corrects multiple sources of overhead arising from microarchitectural properties arising from a research design process and timeline
  - "Benchmark ABI" works around lack of bounds prediction in prototype
  - Morello design on FPGA used to validate other incremental improvements
  - Current best estimate of overhead ceiling presented using "P128" code generation
- "Essential reading" if you are doing any performance work on Morello
- Covered in the next workshop session

Benchmark ABI

- The performance evaluation report shows that capability-relative jumps introduce overhead on Arm Morello in some workloads
  - The overhead is associated with the branch-predictor that was not extended to predict bounds in the Morello prototype
  - This is a Morello prototype performance overhead, not a CHERI overhead – and can be worked around in code generation to better predict future CHERI performance on a mature microarchitecture
- We introduce a new Benchmark ABI that caters to this concern
  - The CheriBSD kernel is aware of the benchmark ABI, and constructs an unbounded PCC for a user-space process
  - A benchmark ABI program is compiled not to use capability-based jumps, leaving PCC unbounded
  - Should not be used for security evaluation; instead use CheriABI compilation that implements bounded PCC

Cornucopia Reloaded: Load Barriers for CHERI Heap Temporal Safety (ASPLOS 2024)

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited

Annona: Load Barriers for CHERI Heap Temporal Safety

Nathaniel Wesley Filardo, Microsoft
Brett F. Gutstein, University of Cambridge
Jonathan Woodruff, University of Cambridge
Jessica Clarke, University of Cambridge
Peter Rugg, University of Cambridge
Brooks Davis, University of Cambridge
Mark Johnston, University of Cambridge
Robert Norton-Wright, Microsoft
David Chisnall, Microsoft
Simon W. Moore, University of Cambridge
Peter G. Neumann, SRI
Robert N. M. Watson, University of Cambridge

Abstract

Violations of temporal memory safety ("use after free", "UAF") continue to pose a significant threat to software security. The CHERI capability architecture has shown promise as a technology for C and C++ language reference integrity and spatial memory safety. Building atop CHERI, prior works – CHERIvoke and Cornucopia – have tantalized heap temporal *safety as well. However, these efforts have sizable CPU and DRAM traffic overheads and significant "stop-the-world" pause times.*

We present Annona, a re-designed drop-in replacement implementation of CHERI temporal safety, using a novel architectural feature – a per-page capability load barrier, added in Arm's Morello prototype core and CHERI-RISC-V – to nearly eliminate application pauses. We analyze the performance of Annona as well as (re-implementations of) Cornucopia and CHERIvoke on Morello, using the CHERI-compatible SPEC CPU2006 INT workloads to assess its impact on batch workloads and using `pgbench` *as a representative interactive workload. We find that Annona achieves its goals: applications no longer experience significant revocation-induced world-stopped periods, the system incurs no additional wall- or CPU-time cost relative to Cornucopia, and, pleasantly, this new approach* reduces *the total DRAM traffic used by revocation by a median of 12% across SPEC CPU2006 benchmarks and by over 50% for* `pgbench`.

1. Introduction

Programming languages, broadly, offer an object-centric (let us not say "oriented") model of memory. New objects, which are unrelated to existing objects, are allocated on demand, used, and then released (implicitly and/or explicitly depending on the language). Lowering the language's model to the underlying architecture, most often built around a coherent, integer-indexed array of memory words, is generally not fully-abstract; it becomes possible to, for example, ...

- confuse integers, object references, and memory indices that do not point to valid objects (such as those used internally by the memory allcoator), risking *reference integrity* violations;
- access adjacent objects, reaching beyond the bounds of a referenced object, violating *spatial safety*;
- access an object after its life ended ("use-after-free", "UAF") or after the underlying memory has been repurposed ("use-after-reallocation", "UAR"), violating *temporal safety*.

These affordances beyond the programmer's intent continue to pose significant threat to software security [11, 25], and a wide variety of languages, compilation approaches, and runtime strategies have emerged in response.

The CHERI [40] capability architecture, summarized in §2.1, has shown promise as a technology for C and C++ language reference integrity and spatial safety, with overheads acceptable for general-purpose computing [39]. Strategies for heap temporal safety atop CHERI have emerged, most notably CHERIvoke [44] and its successor Cornucopia [17], and have hinted at viability of a *sweeping revocation* approach (§2.2). However, while Cornucopia's aggregate overheads may be tolerable for high-security workloads, its sizable application pause times ("stop-the-world" phases) still likely limit its use to *non-interactive* high-security workloads.

Targeting this shortcoming, we exploited recent extensions to the CHERI architecture and built Annona,[1] a drop-in replacement for Cornucopia's in-kernel component. The key architectural feature is a per-page capability load barrier (§3.2), supporting a fast global enablement (§4.1). Annona uses this, in tandem with an improved form of Cornucopia's *capability dirty* tracking (§4.2), to replace Cornu-

[1] Annona was a divine personification of grain supply to Rome and was frequently shown holding a cornucopia.

1

- **Cornucopia heap temporal safety** (IEEE SSP 2020), is a GC-inspired, quarantining technique
  - The kernel virtual-memory subsystem tracks "capability dirty" pages
  - A long "stop-the-world" phase - as much as 30 milliseconds measured in practice
- **Cornucopia Reloaded** (ASPLOS 2024) moves to a GC-inspired "load-barrier"
  - VM invariant is that accessible pages have already undergone revocation
  - Depend on 1-bit capability generation added to VM PTEs, implemented by Morello
  - Stop-the-world pauses 10s of microseconds
- Enabled by default in CheriBSD 23.11

Cornucopia Reloaded in CheriBSD 23.11

- CheriBSD 23.11 will include heap temporal safety enabled by default
  - You can enable, disable it at the system, binary file or process level
- The mrs(3) (Malloc Revocation Shim) man page describes available utilities
  - May also be usable to extend [some] other heap allocators..?
- The default memory allocator in CheriBSD 23.11 is jemalloc
- We are testing it with the GUI stack and other third-party software

Ongoing temporal memory-safety deployment

Looking for increased experience, and appreciate any feedback on:

- Semantic impact on any applications vs. bugs/vulnerabilities discovered
- Acceptability of performance behavior, optimization opportunities
- Use in higher-level allocators – e.g., APR, Chromium, etc.
- Support for strong isolation needed for compartmentalization
- Enabling safe inter-compartment communication via shared memory

Proposed operational models: Isolated libraries and UNIX co-processes

Isolated dynamically linked libraries

- New API loads libraries into in-process sandboxes.
- Calling functions in isolated libraries performs a domain transition, with overheads comparable to function calls.
- Simple model eschews asynchrony, independent debugging, etc.

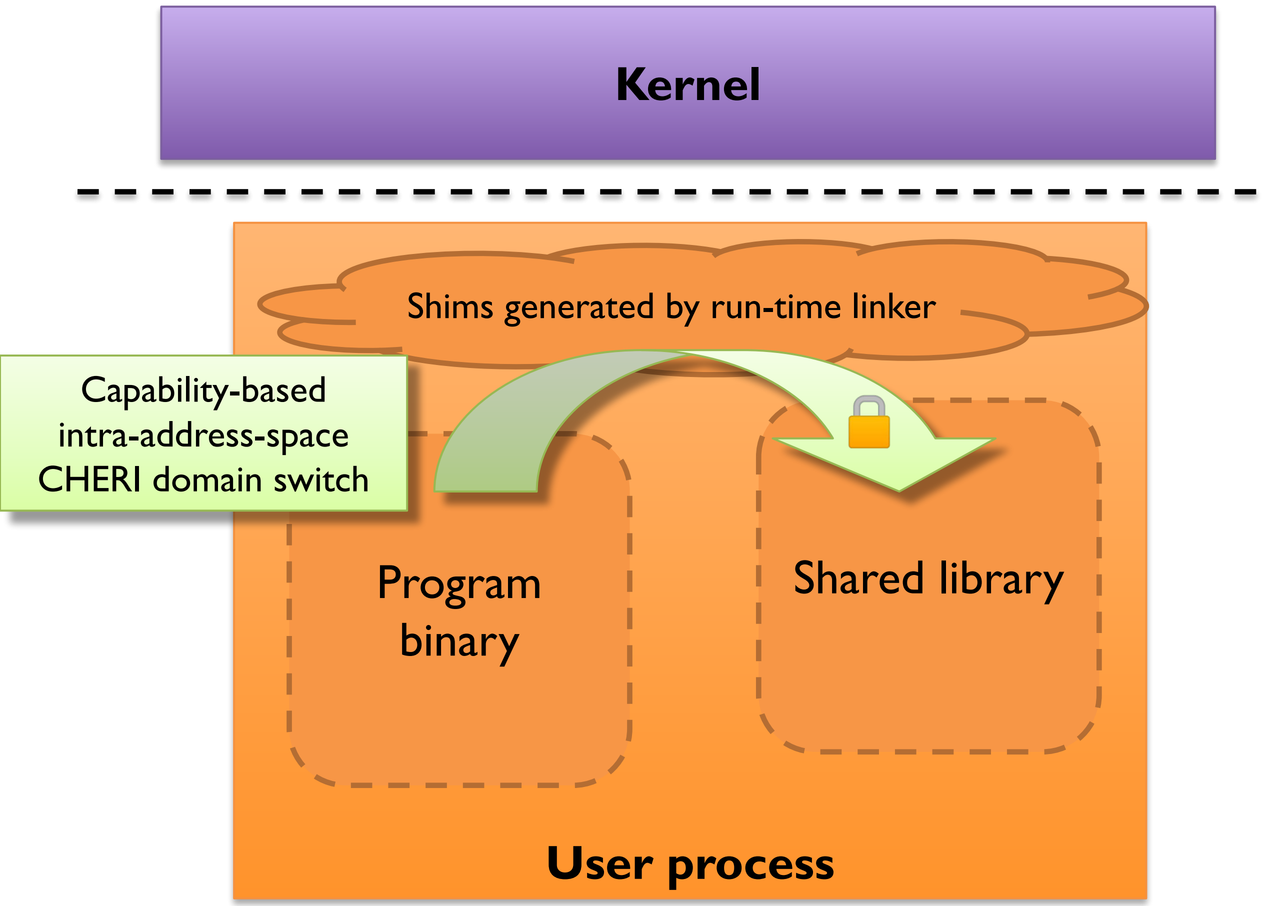
Prototype appeared in CheriBSD 22.12; updates in 23.11

UNIX co-processes

- Multiple processes share a single virtual address space, separated using independent CHERI capability graphs.
- CHERI capabilities enable efficient sharing, domain transition.
- Rich model associates UNIX process with each compartment.

Prototype to appear in future CheriBSD release

Shared library compartmentalization

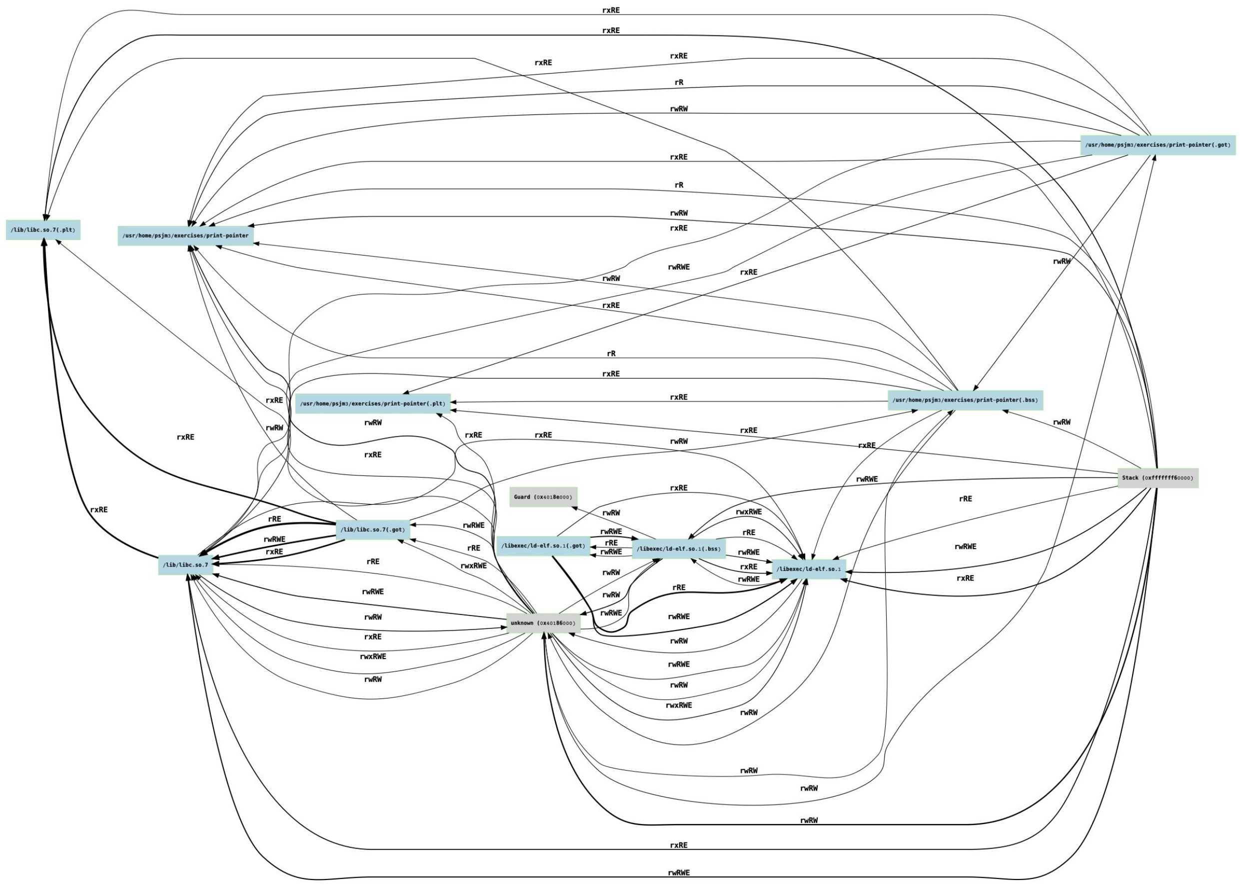


- Run-time linker limits shared libraries to accesses enabled by ELF
  - Adversary model assumes arbitrary code execution within library
  - Run-time linker delegates capabilities for linked functions, globals via GOT/PLT
  - Domain transitions implemented by trampolines interposed on inter-object calls / returns
- Running prototype on Arm Morello
  - Released in CheriBSD 22.12 in December
  - Low measured overheads in early experiments (e.g., ~1% for image decompression sandboxing)
  - More evaluation results for gRPC and nginx at the Capabilities Limited stand

Library-based compartmentalization in CheriBSD 23.11

- (in this workshop) Enabling, tracing, and analysing library compartments
- Changes in 23.11:
  - Support the new ABI for passing memory arguments
  - Trampolines now clear unused argument registers based on function signature information embedded in the ELF. This requires a custom-built toolchain though
  - Allow defining compartments that consist of multiple libraries. Although this functionality is currently only usable via an undocumented interface
  - Support for the benchmark ABI (in testing now – not in today's snapshot)

Capability graph visualization and analysis



- Pointers are now directly visible in hardware – in memory, ISA-level traces, and so on
  - We can directly analyze capability delegation with CHERI
- New extraction tools scan virtual addresses spaces and binaries to enable:
  - Visualization
  - Validation
  - Debugging and optimization
- **Allows direct analysis of attacker-visible resources and attack surfaces**

Capability introspection utilities

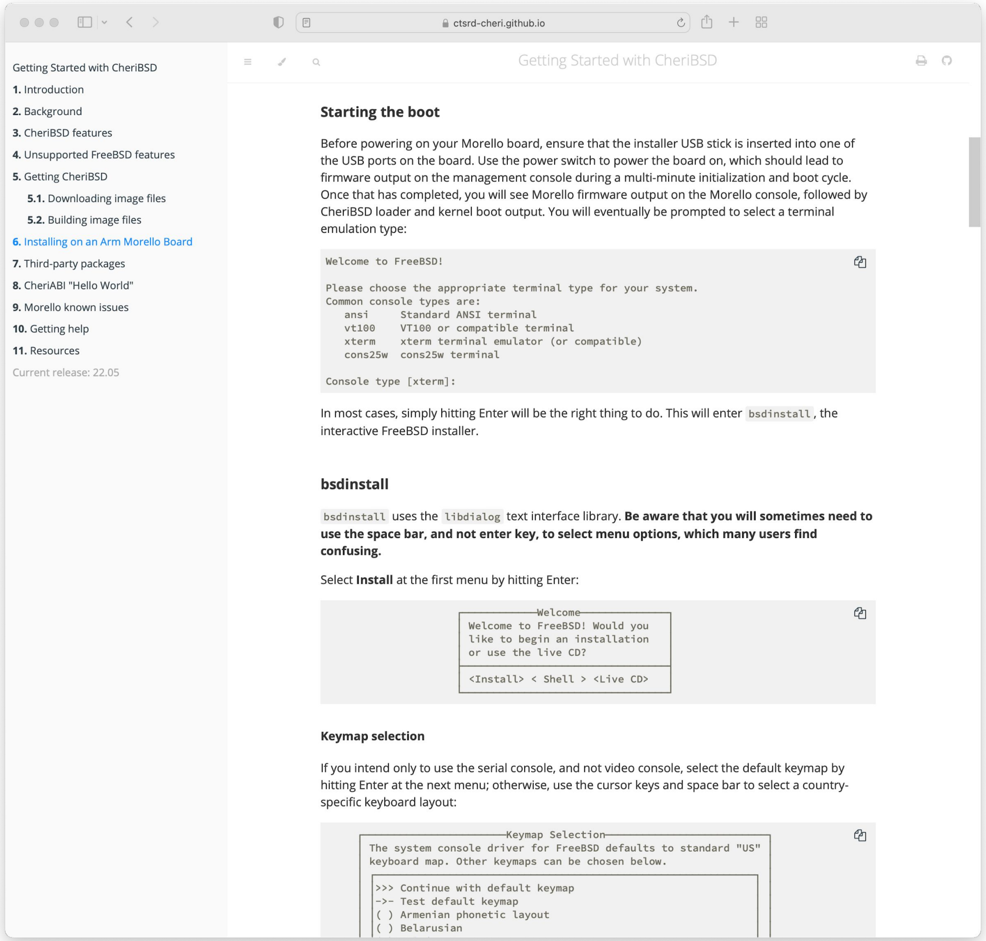
CheriBSD 23.11 will include the chericat utility that collects information about in-memory capabilities and stores them in an SQLite database for analysis and visualization. This tool will improve over time, growing new visualization, analysis, and testing abilities.

CheriTree from rtegrity allows to print a capability derivation tree to assist a developer in applying the library-based compartmentalization model to their application.

Other improvements

- Upstream FreeBSD ports changes from April 2022 up to August, 2023 have been merged into CheriBSD ports
- The number of CheriABI packages increased from 9,104 to 10,307
- Added package manager pkg64cb for Benchmark ABI packages (9,741)
- The experimental co-process compartmentalization model now works with for CheriBSD/Morello – but not yet recommended for use
- bhyve hypervisor changes are being reviewed to be included in FreeBSD (arm64) and then merged into CheriBSD 23.11 (w/CHERI support) this week

Getting Started with CheriBSD **23.11** (WIP)


ctsrd-cheri.github.io

Getting Started with CheriBSD

Getting Started with CheriBSD
1. Introduction
2. Background
3. CheriBSD features
4. Unsupported FreeBSD features
5. Getting CheriBSD
5.1. Downloading image files
5.2. Building image files
6. Installing on an Arm Morello Board
7. Third-party packages
8. CheriABI "Hello World"
9. Morello known issues
10. Getting help
11. Resources
Current release: 22.05

Starting the boot

Before powering on your Morello board, ensure that the installer USB stick is inserted into one of the USB ports on the board. Use the power switch to power the board on, which should lead to firmware output on the management console during a multi-minute initialization and boot cycle. Once that has completed, you will see Morello firmware output on the Morello console, followed by CheriBSD loader and kernel boot output. You will eventually be prompted to select a terminal emulation type:

```
Welcome to FreeBSD!

Please choose the appropriate terminal type for your system.
Common console types are:
   ansi     Standard ANSI terminal
   vt100    VT100 or compatible terminal
   xterm    xterm terminal emulator (or compatible)
   cons25w  cons25w terminal

Console type [xterm]:
```

In most cases, simply hitting Enter will be the right thing to do. This will enter `bsdinstall`, the interactive FreeBSD installer.

bsdinstall

`bsdinstall` uses the `libdialog` text interface library. **Be aware that you will sometimes need to use the space bar, and not enter key, to select menu options, which many users find confusing.**

Select **Install** at the first menu by hitting Enter:

```
Welcome
Welcome to FreeBSD! Would you
like to begin an installation
or use the live CD?

<Install> < Shell > <Live CD>
```

Keymap selection

If you intend only to use the serial console, and not video console, select the default keymap by hitting Enter at the next menu; otherwise, use the cursor keys and space bar to select a country-specific keyboard layout:

```
Keymap Selection
The system console driver for FreeBSD defaults to standard "US"
keyboard map. Other keymaps can be chosen below.

>>> Continue with default keymap
->- Test default keymap
( ) Armenian phonetic layout
( ) Belarusian
```

https://www.cheribsd.org/getting-started/23.11/

- Introduces CheriBSD
- High-level documentation of CHERI-specific features, such as heap temporal safety
- Steps you through installation on a Morello board using a USB stick image that you can download
- Describes third-party package system and pkg64/pkg64c/pkg64cb
- Illustrates "hello world" compilation and debugging for different ABIs – now including Benchmark ABI
- Includes benchmarking guidelines
- Describes some known issues
- Explains how to get support

How to obtain and install the CHERI software stack (1/2)

README.md

`cheribuild.py` - A script to build CHERI-related software (requires Python 3.5.2+)

This script automates all the steps required to build various CHERI-related software. For example `cheribuild.py [options] sdk` will create a SDK that can be used to compile software for the CHERI CPU and `cheribuild.py [options] run-riscv64-purecap` will start an instance of CheriBSD built for RISC-V in QEMU.

`cheribuild.py` also allows building software for Arm's adaption of CHERI, the Morello platform, however not all targets are supported yet.

Supported operating systems

`cheribuild.py` has been tested and should work on FreeBSD 11 and 12. On Linux, Ubuntu 16.04, Ubuntu 18.04 and OpenSUSE Tumbleweed are supported. Ubuntu 14.04 may also work but is no longer tested. macOS 10.14 and newer is also supported.

Pre-Build Setup

macOS

When building on macOS the following packages are required:

```
brew install cmake ninja libarchive git glib automake autoconf coreutils llvm make wget pixman p
# Install samba for shared mounts between host and CheriBSD on QEMU
brew install arichardson/cheri/samba
# If you intend to run the morello FVP model you will also need the following:
brew install homebrew/cask/docker homebrew/cask/xquartz socat dtc
```

Ubuntu

If you are building CHERI on a Debian/Ubuntu-based machine, please install the following packages:

```
apt-get install libtool pkg-config clang bison cmake ninja-build samba flex texinfo libglib2.0-d
```

Older versions of Ubuntu may report errors when trying to install `libarchive-tools`. In this case try using `apt-get install bsdtar` instead.

RHEL/Fedora

If you are building CHERI on a RHEL/Fedora-based machine, please install the following packages:

```
dnf install libtool clang-devel bison cmake ninja-build samba flex texinfo glib2-devel pixman-de
```

Basic usage

If you want to start up a QEMU VM running CheriBSD run `cheribuild.py run-riscv64-purecap -d` (-d means

- One build tool to rule them all: cheribuild

  https://github.com/CTSRD-CHERI/cheribuild

- Builds, installs, and/or runs:
  - QEMU CHERI-RISC-V and Morello, Morello FVP
  - CheriBSD/CHERI-RISC-V and Morello disk images
  - Small suite of adapted third-party applications
- Up and running with one command:

  ./cheribuild.py --include-dependencies run-riscv64-purecap

  ./cheribuild.py --include-dependencies run-morello-purecap

How to obtain and install the CHERI software stack (2/2)

You can also bootstrap a Docker container with CheriBSD for CHERI-RISC-V or Arm Morello using:

docker pull ctsrd/cheribsd-sdk-qemu-riscv64-purecap

docker pull ctsrd/cheribsd-sdk-qemu-morello-purecap

Part II: practice

Exercises.

The CheriBSD 23.11 tutorial is available at:

cheribsd.org/tutorial/23.11/

Instructions and exercises can be found there now

The jails will remain active until November 15.

Work environment

- You will work with CheriBSD/Morello using a FreeBSD jail (container)
- You only need an SSH client
- There are pre-installed packages:
  - CheriABI: git, nano, chericat, tmux
  - Benchmark ABI: pkg64cb
  - Hybrid ABI: llvm-base, gdb-cheri, vim
- You can switch to **root** with **sudo su -l**
  You can install packages and change the configuration of your jail
- You can install your own SSH key in ~/.ssh/authorized_keys
- We will distribute initial SSH credentials on paper

Talk to us on the CHERI-CPU Slack or mailing lists:

https://cheri-cpu.org/

(→ CHERI → CHERI-CPU Slack)

To find get started on the practical material:

cheribsd.org/tutorial/23.11/