A Dozen Years of CheriBSD

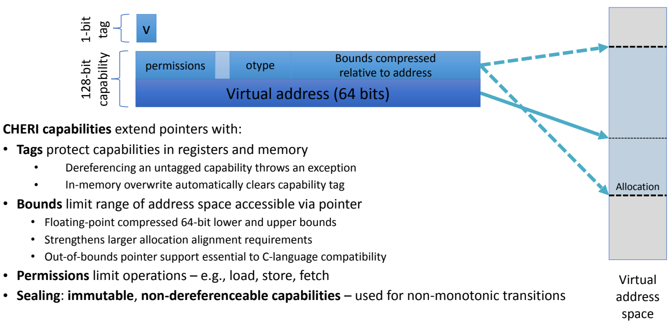
BY BROOKS DAVIS

Since late 2010, the CHERI research project at the University of Cambridge and SRI International has striven to develop, demonstrate, and transition to real-world products architectural extensions providing memory safety and efficient compartmentalization. CheriBSD, our CHERI-enhanced fork of FreeBSD, is one of the most important products of our work. Adapting FreeBSD to support CHERI has informed our architectural changes while demonstrating that our ideas can work at the scale of a large modern operating system.

A Brief Introduction to CHERI

CHERI extends existing architectures (Armv8-A, MIPS64 (retired), RISC-V, and x86_64 (in development)) with a new hardware type, the CHERI capability. In CHERI systems, all access to memory is via CHERI capabilities either explicitly via new instructions or implicitly via a Default Data Capability (DDC) and Program Counter Capability (PCC) used by instructions with integer arguments. Capabilities grant access to specific ranges of (virtual, or occasionally, physical) memory via a base and length, and can further restrict access with permissions, which are compressed into a 128-bit representation (64-bits for the address and 64-bits for the metadata). In memory and in registers, capabilities are protected by tags that are cleared when the capability data is modified by a non-capability instruction or if a capability instruction would increase the access the capability grants. Tags are stored separately from data and cannot be manipulated directly.

CHERI 128-bit capabilities



Our initial work on CHERI extended the MIPS64 architecture as part of the DARPA CRASH program. In 2014 we began collaboration with Arm, exploring the possibility of adapting CHERI to the Armv8-A architecture. In 2017 we began a port of CHERI to RISC-V informed by both our MIPS work and our collaboration with Arm. This port was performed as part of the DARPA MTO SSITH program. Our collaboration with Arm became public in 2019, with the announcement of the £190m Digital Security by Design program, which has resulted in the Morello architecture prototype, a SoC based on the Neoverse N1 core used in cloud platforms such as Amazon Web Services' Graviton nodes.

We have designed CHERI capabilities to be suitable for use as C and C++ language pointers and have modified the Clang compiler to support them in two modes. In hybrid mode, pointers annotated with `__capability` are capabilities, while other pointers remain integers. In pure-capability mode, all pointers are capabilities, including implied pointers such as return addresses on the stack. Coupled with kernel support and modest changes to the C startup code, run-time linker, and standard library, we have produced a memory safe C/C++ runtime environment called CheriABI[1]. The refinement of this environment is a key thrust of our work on CheriBSD alongside creation of a pure-capability kernel environment and explorations of temporal memory safety and compartmentalization.

In addition to memory safety, CHERI enables fine-grained compartmentalization. Because all memory accesses are via capabilities, the portion of an address space a given thread can reach is defined by its register set and the memory that can be (transitively) reached from there. With appropriate mechanisms to transition between register sets, we can switch rapidly among compartments. Various CHERI implementations implement different mechanisms for this; which one(s) are most appropriate to a commercial implementation remains the subject of active research.

What is CheriBSD?

CheriBSD is FreeBSD modified to support CHERI. But what does that actually mean?

When the kernel is compiled for CHERI, the default ABI is a pure-capability ABI (CheriABI) where all pointers including system-call arguments are capabilities. We also support both hybrid binaries and standard FreeBSD binaries via the freebsd64 ABI compatibility layer derived from the freebsd32 32-bit compatibility layer. Likewise, we build libraries, programs, and the run-time linker for CheriABI by default and build libraries for hybrid binaries that are installed in `/usr/lib64` just like `/usr/lib32` for freebsd32. All of this means that by default users are presented with a memory-safe Unix userland which retains the ability to run unmodified FreeBSD binaries.

The kernel can be compiled as either a hybrid or a pure-capability program. This adds some complexity to the changes we need to make (every pointer to userspace requires an annotation (`__capability`) for hybrid), but we started out with hybrid in the early days of the project when we didn't have strong C compiler support, and pure-capability kernels do have somewhat higher inherent overhead due to the increased pointer size. All internal kernel development is done with pure-capability support in mind. This work includes ensuring that all access to userspace is via a capability[2], changes to the VM system to create capabilities when allocating memory and altering device drivers including the DRM GPU framework to use capabilities.

Historically, CheriBSD has mostly been a compile-from-source proposition. This is familiar to FreeBSD developers, and has many benefits; however, for people who just want to port a custom codebase to CHERI, that's a big hurdle. With the release of Arm's Morello prototype, we've started producing full releases with an installer and packages. We use a lightly customized version of the FreeBSD installer that adds support for installing a GUI desktop environment based on KDE and removes some dialog boxes we deemed confusing. The GUI environment is comprised of packages built from our fork[3] of the FreeBSD ports collection. Because not all software has been ported to CHERI, we build two sets of packages and build and install two versions of the `pkg` command with `pkg` being a script that redirects callers to the other names. There is a CheriABI set which is managed by the `pkg64c` command and installed under `/usr/local` and a hybrid set managed by the `pkg64` command and installed under `/usr/local64`. Most of the desktop environment is CheriABI binaries, with the big exception being web browsers (a CheriABI port of Chromium is in progress). Post-install, hybrid packages are also useful for installing not-yet ported software such as emacs and Morello LLVM.

Beyond memory safety, CheriBSD plays host to much of our research on software compartmentalization. In the MIPS era, we implemented a compartmentalization framework (libcheri) that we applied to the integrated version of tcpdump. While we did not port this work forward to RISC-V and Morello, it informed our early thinking on the use of compartmentalization for increased availability. Our latest release contains a library compartmentalization model where the dynamically linked library runs in its own sandbox. The current implementation is experimental but shows considerable promise at compartmentalizing programs with little or no modification. Additionally, in a stack of development branches, we have a co-process compartmentalization model in which multiple processes share the same virtual address space, relying on CHERI to provide memory isolation. Coupled with a trusted switcher component, this enables extremely fast transition of execution from a thread in one process to a thread in another process. We expect a signification portion of future work on CheriBSD will be motivated by compartmentalization, as we refine our models in the face of an increasing corpus of compartmentalized software.

CheriBSD is both a research artifact under active development and a product servicing dozens or hundreds of users doing their own R&D. Even for users targeting other domains (embedded systems, Linux, Windows, etc.) CheriBSD is currently the easiest place to test CHERI technologies.

Why CheriBSD?

Historically hardware research has focused on bare metal benchmarks or embedded operating systems. They have a lower memory footprint and usually execute fewer instructions (important for simulation) as well as simply having less code to understand and change. Unfortunately, results don't always scale to real world operating systems and it's too easy to hand wave at things like dynamic linking as "a small matter of programming." Adapting FreeBSD was undeniably more work, but doing so has given us the ability to evaluate CHERI with an unmatched level of realism. Some of our ability to use a real, multi-user operating system stems from timing. In 2010, FPGAs big enough to run simple cores supporting full instruction set architectures at decent speeds (100MHz) were finally available for reasonable prices ($5-10k vs $100k or much more). Likewise, desktop computers were big enough and fast enough to support full system emulators like QEMU with relative ease.

People do ask: "why not Linux?" FreeBSD offers a number of advantages for a research project like CHERI. On the technical front, FreeBSD's integrated build system and early adoption of LLVM has made it relatively easy to build large corpuses of software with experimental compilers (C/C++ compiler research is mostly done in LLVM today) both in the base system and via the ports tree. The clean ABI (Application Binary Interface) abstractions to support Linux binaries and the freebsd32 32-bit compatibility layer greatly simply ABI experimentation. (By contrast, Linux supports a single alternative ABI that must be 32-bit, and Windows does all the translation within userspace via a DLL.) While not part of our initial decision, it later emerged that choosing FreeBSD over Linux was fortuitous due to extensive use of `long` in the Linux kernel for both integers and pointers, which cause capabilities to be invalidated. While people are working on Linux ports at Arm and elsewhere, the use of `long` is a major stumbling block.

Historically hardware research has focused on bare metal benchmarks or embedded operating systems.

On less technical fronts, BSD and FreeBSD have a long history of successful research and transition to real-world products. From the Fast File System (FFS) and sockets APIs for TCP/IP in 4.2BSD to Capsicum and pluggable TCP/IP stacks in FreeBSD, many ideas in daily use by billions of people have been incubated in BSD. One factor in this success is FreeBSD's permissive license. Publishing our work under the two-clause BSD license means potential adopters can easily evaluate our work even within companies with proprietary operating systems and strict controls around GPL-licensed software. This has enabled successes like a very positive evaluation[4] of past Windows security vulnerabilities by the Microsoft Security Response Center.

Ultimately, the success of CHERI depends on adoption by multiple operating systems. Today, CheriBSD leads the pack with the latest features and most active research.

A CheriBSD Timeline

- October 2010—The first CHERI Project begins
- May 2012—CheriBSD running on CHERI-MIPS CPU.
- November 2012—Sandboxed custom application demo on CheriBSD.
- October 2013—Migrated development to git.
- January 2014—CheriBSD compiled with CHERI LLVM.
- November 2014—Sandboxed tcpdump (sandbox per-decoder).
- June 2015—CheriBSD with compressed capabilities (128-bit vs 256-bit).
- September 2015—CheriABI pure-capability process environment up and running.
- January 2016—Began merging RISC-V support from FreeBSD.

There are over 1800 commits to the FreeBSD source tree with "Sponsored by:" lines indicating they were likely funded by work on CHERI.

- April 2019—CheriABI paper wins Best Paper award at ASPLOS 2019.
- September 2019—Morello CPU, SoC, and board announced.
- August 2020—CheriBSD ported to CHERI-RISC-V.
- June 2021—Pure-capability kernel (RISC-V)
- January 2022—First official Morello boards ship. CheriBSD aided in validation.
- May 2022—CheriBSD 22.05 release targets Morello board users. This is an initial support release focusing on the installer and basic package infrastructure. The package set included a basic set of tools including the Morello LLVM compiler.
- December 2022—CheriBSD 22.12 release includes library-based compartmentalization, ZFS support, DRM support for the on-die GPU, and a basic GUI environment where everything except the web browsers is a pure-capability program.

Benefits to FreeBSD

Research projects like CHERI can provide significant benefits to FreeBSD. We have contributed changes ranging from typo fixes to a port to the RISC-V architecture. We've also given talks, added new committers, and introduced many organizations to FreeBSD.

There are over 1800 commits to the FreeBSD source tree with "Sponsored by:" lines indicating they were likely funded by work on CHERI[5]. This amounts to over 1.5% of commits outside `contrib` and `sys/contrib` since January 2011. These contributions have been made possible by funding over a dozen committers so far including two new ones.

Notable contributions:

- External toolchain support—I contributed initial support, later enhanced by Baptiste Daroussin to add the `CROSS_TOOLCHAIN` variable used today. This functionality was added to support compiling with the CHERI Clang compiler as well as custom compilers developed for two other projects: TESLA and SOAAP. TESLA enabled construction and dynamic enforcement of temporal logic assertions, and SOAAP allowed exploration of compartmentalization hypotheses for large applications.
- Unprivileged installs and images—I ported the ability to store the owner and permission metadata of installed files in a `METALOG` file from NetBSD in January 2012. This allows the `intallworld` command to be run without root privileges. Coupled with support in `makefs` it was then possible to build UFS filesystems of either endianness. Followed by my complaints that there wasn't a way to embed a filesystem in a partition table without mounting it, Marcel Moolenaar contributed the `mkimg` command in March 2014 to complete the required toolking.
- MIPS64 maintenance—While FreeBSD had a MIPS port (essential for our use), it didn't have a lot of users, and didn't get much maintenance. We did quite a bit to keep it running, and improved things that hit our pain points. It served us well, but we breathed a sigh of relief when we'd transitioned our last work to RISC-V and MIPS was removed from the main branch.
- RISC-V port—While MIPS had served us well, and we were trying to build a community around our base BERI MIPS FPGA implementation, it become clear that the research community was moving to RISC-V. As a result, we tasked Ruslan Bukin with porting FreeBSD to RISC-V; he landed it in the tree in January 2016.
- Arm N1SDP platform support—The Morello platform is based on Arm's N1SDP development board. Ruslan worked with Andrew Turner to support the attached peripherals, including the PCI root complex and IOMMU in 2020.
- Cross build from macOS and Linux—In September 2020, Alex Richardson contributed a make wrapper (`tools/build/make.py`) that allows `bmake` and other build tools to be bootstrapped on a non-FreeBSD system. This allows builds on users' non-FreeBSD desktops and laptops, and in CI environments that don't support FreeBSD. Alex and Jessica Clarke maintain this support on an ongoing basis.
- Consolidated compatibility system call stubs—Historically, system calls have been declared in `sys/kern/syscalls.master` with compatibility versions declared in `sys/compat/freebsd32/syscalls.master`. Developers would fail to keep them in sync or misunderstand if they needed a compatibility wrapper. As part of adding two ABIs to CheriBSD, I extended the `syscalls.master` file format and stub generation code with enough understanding of ABIs for the script to know what is required. Now there is only one list of system calls and freebsd32 has a `syscalls.conf` that specifies ABI details. I upstreamed this work in early 2022.
- Unprivileged, cross release builds—As part of supporting hundreds of users of Morello hardware we needed to start producing releases. Most of our CI and build infrastructure does unprivileged builds on Linux hosts so Jessica closed the last gaps in unprivileged builds and cross build support allowing us to build release images in February 2022.

In addition to these changes, we've made many smaller improvements along the way. With over 1,800 commit messages, I'd use up all my word count use listing a fraction of them.

Beyond technical contributions, the CHERI project has contributed to the community. We've added two new committers: Alexander Richardson and Jessica Clarke. We've also had contributions from graduate students including Alfredo Mazzinghi and Dapeng Gao. From short-term contracts to full-time employment, at one time or another we've supported committers including: Jonathan Anderson, John Baldwin, Ruslan Bukin, David Chisnall, Jessica Clarke, Brooks Davis, Mark Johnston, Ed Maste, Edward Napierala, George Neville-Neil, Philip Paepes, Alexander Richardson, Hans Petter Selasky, Stacey Son, Andrew Turner, Robert Watson, Konrad Witaszczyk, and Bjoern Zeeb

Further, we've exposed many people to FreeBSD as a research platform. We've been part of three DARPA programs (CRASH and MRC from the I2O program office and SSITH from MTO) where people gained FreeBSD experience as part of supporting and evaluating our work. With the UK Digital Security by Design program, dozens of organizations are now using CheriBSD in demonstration projects funded by Digital Catapult and the Defence Science and Technology Laboratory (DSTL).

Conclusions

As research projects go, CHERI has been enormously successful, and FreeBSD has played a major role in that success. Having a well-integrated base OS and monolithic build system, coupled with the ports collection's massive scale, has allowed us to demonstrate CHERI's potential to a wide audience—leading to real-world implementations ranging from Arm's server-class Morello design to Microsoft's CHERIoT microcontroller. In turn, CheriBSD development has led to significant improvements in FreeBSD from the RISC-V port to build system improvements.

Footnotes

1. https://www.cl.cam.ac.uk/research/security/ctsrd/pdfs/201904-asplos-cheriabi.pdf
2. A few subsystems access userspace via the direct map, and those are validated rather than using capabilities directly.
3. https://github.com/CTSRD-CHERI/cheribsd-ports
4. https://msrc-blog.microsoft.com/2020/10/14/security-analysis-of-cheri-isa/
5. A portion of lines matching "Sponsored by:.*DARPA" are from the CADETS project which focused on Dtrace work, but the vast majority are CHERI related.

BROOKS DAVIS is a Principal Computer Scientist in the Computer Science Laboratory at SRI International and a Visiting Research Fellow at the University of Cambridge Department of Computer Science and Technology (Computer Laboratory). Leads development of CheriBSD, a fork of FreeBSD supporting CHERI ISA extensions. He has been a FreeBSD user since 1994, a FreeBSD committer since 2001, and has served 4 terms on the core team.