CheriBSD Ports and Packages

Pure-capability third-party software for Arm Morello and CHERI-RISC-V CheriBSD

BY KONRAD WITASZCZYK

CHERI is a hardware/software/semantics co-design project that aims to improve the security of existing and future hardware-software stack implementations. Recent studies from Google and Microsoft show that around 70% of vulnerabilities in their products relate to memory-safety issues. CHERI not only allows us to prevent most such vulnerabilities from being exploited but also to compartmentalise software and thus limit the impact of successfully exploited vulnerabilities currently unknown to software maintainers (e.g., backdoors in third-party software dependencies).

Until 2022, the CHERI project was mostly developed by the University of Cambridge, SRI International, and their partners, including Microsoft, Google, and Arm. The CHERI ecosystem rapidly expanded with the release of the Arm Morello platform, the first public hardware implementation of CHERI. In January 2022, Arm started shipping the first (out of roughly a thousand) Morello boards to companies, academic and government institutions. To provide a user-friendly work environment for Morello users, *CheriBSD* — a FreeBSD-based operating system adapted for Arm Morello and CHERI-RISC-V — needed an infrastructure to build and ship CHERI-adapted third-party software before Morello was released. Today, dozens of universities, government research labs, and companies are using CheriBSD in their work on Morello, and rely on this infrastructure daily.

The CHERI ecosystem rapidly expanded with the release of the Arm Morello platform.

This article describes our journey of building third-party software packages for CheriBSD without CHERI-enabled hardware, using QEMU user mode, FreeBSD ports, and Poudriere. While discussing implementation details of the package building infrastructure, the article summarises what decisions and changes we needed to make to finally achieve ~24,000 AArch64 packages and ~9,000 CHERI-enabled packages.

CHERI Hardware-software Stack

In order to fully understand the infrastructure for CheriBSD package building, we should first describe the SDK that a developer can use to build software for CHERI. The CHERI hardware-software stack (see Table 1) consists of hardware, emulators, compilers, debuggers, operating systems and applications for CHERI-enabled operating systems. Each component of this stack needed to be adapted for CHERI and must implement support for CHERI capabilities[22].

Third-party software	~9,000 CHERI packages (Morello) ~24,000 non-CHERI packages (Morello)
Operating systems	CheriBSD (Morello, CHERI-RISC-V) FreeRTOS (CHERI-RISC-V) CHERIoT RTOS (CHERI-RISC-V) Linux (Morello) Android (Morello)
Toolchains	CHERI LLVM for CHERI C/C++ (Morello, CHERI-RISC-V) Morello GCC for CHERI C/C++ (Morello) GDB-CHERI (Morello, CHERI-RISC-V)
CPUs	Arm Morello SoC CHERI-RISC-V on FPGA QEMU-CHERI (Morello, CHERI-RISC-V) Microsoft CHERIoT (CHERI-RISC-V)

Table 1: Current CHERI hardware-software stack

Before the Arm Morello platform[20] was released, CheriBSD and third-party software had been developed and ported using QEMU emulators for Morello and CHERI-RISC-V[9]. This environment is still useful today to work on multiple CheriBSD branches or to attach the GDB debugger to QEMU and step through the CheriBSD kernel. Anyone interested in our research can try the CHERI exercises[25] to explore under QEMU how CHERI prevents memory-safety issues. A Morello QEMU-based VM can be created on FreeBSD, Linux and macOS with the *cheribuild* utility[1] using one simple command that fetches and compiles required software, and runs the VM:

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

Available toolchains include LLVM compilers[14, 17] and GDB debuggers[15]. LLVM can cross-compile code or compile it natively on hardware or under QEMU. GDB-CHERI, currently based on GDB 12, can disassemble capability-aware instructions and print information on register and in-memory capabilities. While this article focuses on CheriBSD[3], Arm also develops Linux and Android operating systems[18] with CHERI LLVM and GCC compilers for Morello[19]. In February 2023, Microsoft also published the CHERIoT project[16] that implements a full hardware-software stack with the CHERIoT RTOS for embedded RISC-V devices.

Having the above SDK, we decided to fork FreeBSD ports and extend them with bug fixes and changes necessary for CHERI and CheriBSD. We call this ports collection *CheriBSD ports*.

The process of porting software to CHERI is similar to porting code developed for 32-bit architectures to 64-bit architectures. A pure-capability program can only use CHERI capabilities and CHERI-aware CPU instructions to access memory. A pointer in such a program

has its size increased to 128 bits to hold a CHERI capability. In order to compile a C/C++ program, the code must be adapted to the CHERI C/C++ semantics[24] that require the use of appropriate data types to store pointers (e.g., uintptr_t instead of long), and increase the alignment of pointers to 16 bytes. CHERI LLVM can identify many incompatibilities between C/C++ and CHERI/C++, and display detailed warnings suggesting what changes should be applied in code to make it compatible with CHERI. In many cases, a developer can successfully compile and run their software after fixing all issues found by CHERI LLVM. However, extensive testing is recommended to make sure that ported software does not include any run-time bugs (e.g., misaligned allocations in a custom memory allocator).

While a lot of open-source projects have been ported to CHERI[23], many crucial applications still cannot be compiled to use CHERI capabilities. For example, web browsers are extremely complicated pieces of software requiring lots of dependencies. In order to provide a fully functional development platform, CheriBSD allows to run both CHERI-adapted applications and applications compiled for a baseline architecture of a CHERI-extended CPU (e.g., Armv8-A for Morello). Compatibility with existing software has been essential to the CHERI project to allow to incrementally adapt software for CHERI rather than require to reimplement an application from scratch. FreeBSD, as a baseline operating system for CheriBSD, enabled the implementation of run-time environments for legacy and CHERI-aware software. However, when it comes to providing third-party software for multiple run-time environments, there are still some challenges that CheriBSD inherited from FreeBSD.

Compatibility with existing software has been essential to the CHERI project to allow to incrementally adapt software for CHERI.

Multi-ABI support

FreeBSD includes a feature called compatibility layers that provides system call implementations for programs compiled for different ABIs than the native ABI. For example, an amd64 FreeBSD kernel with a compiled-in 32-bit compatibility layer (also known as *freebsd32*) can run a program compiled for i386. CheriBSD benefits from this feature and supports two ABIs relevant to CHERI: CheriABI also known as the pure-capability ABI (`MACHINE_ARCH` aarch64c and riscv64c) for programs that can only use CHERI capabilities to access memory, and the hybrid ABI (`MACHINE_ARCH` aarch64 and riscv64) for programs that can but do not have to use CHERI capabilities. The latter ABI is implemented by the pure-capability CheriBSD kernel with the *freebsd64* compatibility layer, similar to freebsd32.

Missing cross-ABI support

While the FreeBSD and CheriBSD kernels implement support for multiple ABIs, multi-ABI environments are not supported by FreeBSD ports and Poudriere. This is a major issue in the context of CHERI. Many ports require dependencies that have not been adapted for CHERI yet. For example, Meson and Ninja are commonly used build systems depending on Python. Since we do not have CheriABI Python at the moment, we cannot build these utilities for CheriABI to compile other ports. If FreeBSD ports and Poudriere supported compile-time cross-ABI dependencies, we could use hybrid ABI Meson and Ninja to build CheriABI packages that do not require them at run time. The *CheriBSD ports* section briefly explains how we managed to partially resolve this issue.

Package manager(s)

The **pkg(8)** package manager can only manage packages built for one ABI — by default the ABI of the base system (based on **uname(1)**). For example, i386 packages cannot be installed on an amd64 host alongside amd64 packages and be registered in a single package database (**pkg-register(8)**). Of course, it is possible to create a package with binaries and shared libraries compiled for i386 that is marked as created for amd64, just like FreeBSD does for Linux packages, but that would require creating two packages for the same port (for amd64 and i386), and does not reflect the actual ABI of packaged files at the package manager level. There are two important issues that would have to be resolved to better support such multi-ABI environment:

1. Two packages with the same pre-compiled port but for different ABIs must use distinct paths not to conflict with each other.
   For example, Git compiled for two different ABIs with the same local base path (e.g., **/usr/local**) would conflict on files installed within that path (e.g., **/usr/local/bin/git**).
2. A package for one ABI should be able to depend on a package for a different ABI.
   For example, Git depends on Perl because it includes multiple Perl scripts used by its subcommands (e.g., **git** add **-i**). Instead of using an interpreter compiled for the same ABI, it could use Perl available for another supported ABI.

As of today, we solved the first issue and decided to ignore the second one for CheriBSD.

Not to create conflicts between packages in CheriBSD, we place CheriABI and hybrid ABI packages in two separate locations. We build CheriBSD ports for CheriABI with **LOCALBASE** set to **/usr/local** and hybrid ABI packages with **LOCALBASE** set to **/usr/local64**. While the FreeBSD ports build system provides the localbase feature (in **Mk/Uses/localbase**), we discovered and fixed many ports that break this functionality, e.g. by hardcoding paths in their code; or not using the localbase feature at all.

Built packages are registered in two separate package repositories that can be managed with separate package managers: **pkg64c** for CheriABI packages, and **pkg64** for hybrid ABI packages. **pkg64c** and **pkg64** are programs compiled for the same ABI as packages they manage, they use separate package repository configuration directories, databases and caches. In short, the package managers are completely unaware of each other.

CheriBSD ABI version

By default, the **pkg(8)** package manager decides which package repository to use based on the **NT_FREEBSD_ABI_TAG** ELF note of **uname(1)**. The value of that note is used to construct a value of the ABI pkg variable that can be embedded in a package repository URL (see **pkg.conf(5)** and **/etc/pkg/FreeBSD.conf**). For example, the URL:

```
pkg+http://pkg.FreeBSD.org/${ABI}/latest
```

is expanded to:

```
pkg+http://pkg.FreeBSD.org/FreeBSD:14:amd64/latest
```

on an amd64 host running FreeBSD 14-CURRENT.

In contrast to FreeBSD, CheriBSD does not have any assumptions regarding ABI stability across its releases and branches. Instead, CheriBSD maintains the ABI counter **__CheriBSD_version** (set to the current date as it is bumped), similar to **__FreeBSD_version** and also in **sys/param.h**, that describes the current ABI version used by a CheriBSD branch. In re-

sult, two CheriBSD releases can use the same ABI version and two different CheriBSD branch revisions can use two distinct ABI versions.

This approach allows us to flexibly make changes in the CheriBSD development branch and provide package repositories to users using different revisions of this branch. As for releases, we do not make any changes that would break the ABI within a single release as it would heavily disrupt users' work environments and require recompiling all user code.

We extended the *csu* code in CheriBSD to include the `__CheriBSD_version` counter in the additional `NT_CHERIBSD_ABI_TAG` ELF note of each program compiled for a given branch. Instead of using `NT_FREEBSD_ABI_TAG`, `pkg64` and `pkg64c` use `NT_CHERIBSD_ABI_TAG` when building an URL to a package repository. For example, the URL:

```
pkg+http://pkg.CheriBSD.org/${ABI}
```

is expanded by `pkg64c` to:

```
pkg+http://pkg.CheriBSD.org/CheriBSD:20220828:aarch64c
```

and by `pkg64` to:

```
pkg+http://pkg.CheriBSD.org/CheriBSD:20220828:aarch64
```

on a Morello host running CheriBSD 22.12.

Package building

The CheriBSD/Morello package building infrastructure consists of a local machine starting a build, a FreeBSD/amd64 host building CheriABI packages using the QEMU user mode and a FreeBSD/arm64 host building hybrid ABI packages natively. The builders use the following software stack:

- QEMU BSD user mode for CheriABI programs[10];
- CheriBSD base system;
- CHERI LLVM toolchain;
- CheriBSD ports[6];
- Poudriere extended for CheriBSD[5, 7];
- Poudriere configuration files and helper scripts (e.g., `poudriere-remote.sh`)[8].

Figure 1 presents an overview of the above components. Upon a command from `poudriere-remote.sh`, the FreeBSD/amd64 and FreeBSD/arm64 hosts create Poudriere jails, ports trees, and build the ports trees in CheriBSD/aarch64c and CheriBSD/aarch64 jails, respectively. The CheriBSD/aarch64c jails execute programs compiled for CheriABI using the QEMU user mode while toolchain utilities compiled for the amd64 architecture are executed natively. Similarly, the CheriBSD/aarch64 jails execute all programs natively as they are compiled for arm64. There are currently no ports' hybrid ABI compile-time dependencies that partially use CHERI capabilities and must be executed during the building process; thus, no QEMU user mode is needed for the hybrid ABI packages. The following sections describe the building infrastructure components in more detail.

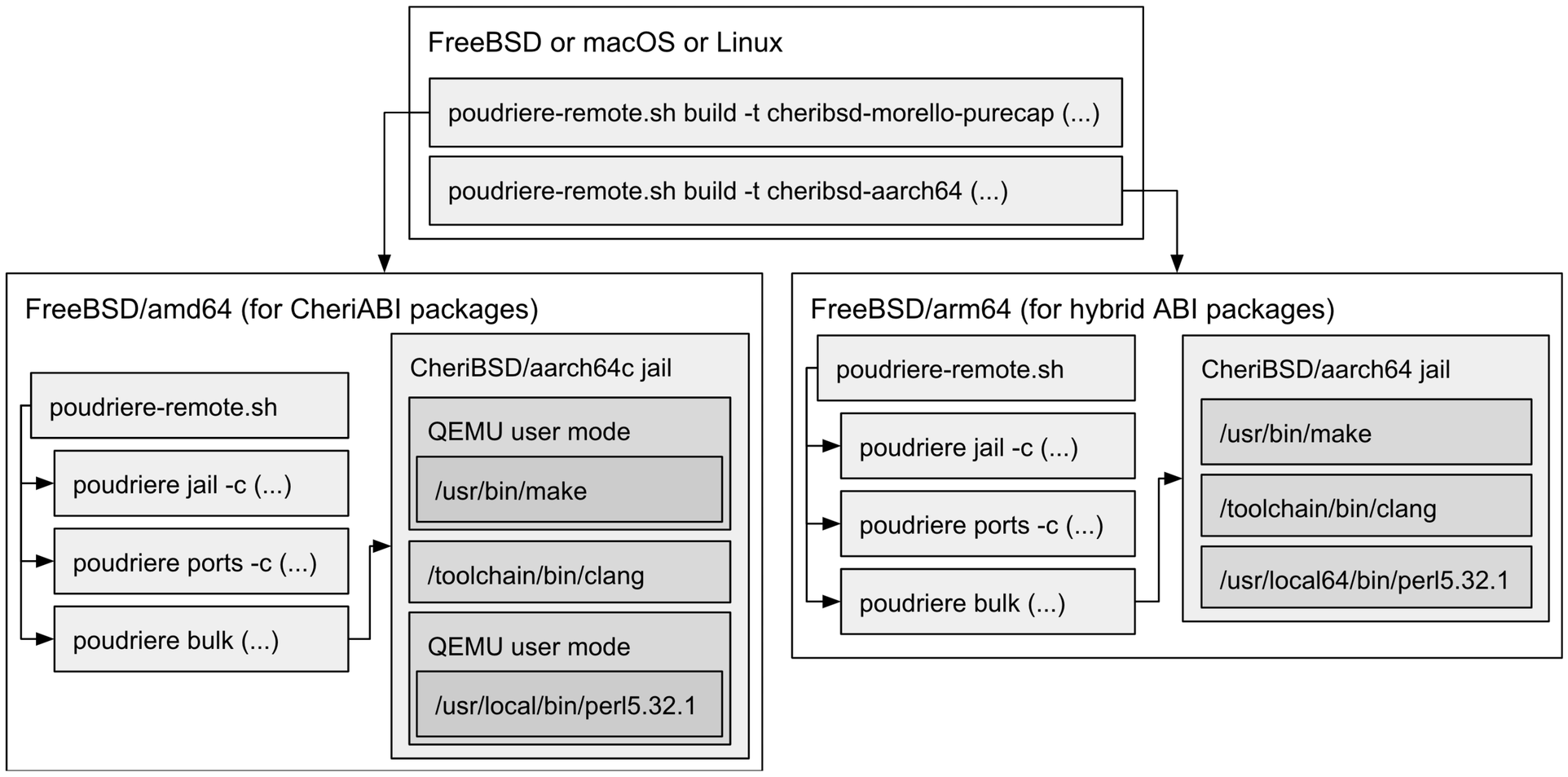

Figure 1: Package building process for CheriBSD

QEMU BSD user mode

Before the package building project started, QEMU-CHERI implemented only the QEMU system mode for CHERI-RISC-V and Arm Morello architectures. While the system mode allows developers to experiment with CheriBSD and cross-compiled third-party software, its performance is not sufficient to build large code scope projects because it emulates a full operating system with devices. Thankfully, Poudriere implements support for the QEMU user mode, on top of **`binmiscctl(8)`**. The user mode emulates user program instructions and executes system calls by translating them from their emulated user versions to native user versions, executing the translated system calls and translating results back to their emulated versions. Using this mode, we can run processes without unnecessary overhead related to system emulation. Most system calls in CheriBSD are compatible with FreeBSD and QEMU can handle any incompatibilities when translating them (e.g., when handling an **`mmap(2)`** call, an allocation might need to be padded with guard pages to make a returned CHERI capability representable[24]). However, we must make sure that a FreeBSD host running the user mode is not older than a baseline FreeBSD version used by the base system of an emulated CheriBSD branch.

Poudriere makes use of the user mode through the *imgact_binmisc* kernel module. FreeBSD allows defining binary image activators with **`binmiscctl(8)`** that execute binaries matching an ELF header pattern using a specific interpreter. For example, a system administrator can define an activator that runs an aarch64 binary using a QEMU user mode emulator on an amd64 host. In practice, a program executed within a FreeBSD/aarch64 jail on a FreeBSD/amd64 host is wrapped with the user mode, e.g.:

```
$ sh
```

is executed within the jail as the command:

```
$ /usr/local/bin/qemu-aarch64-static sh
```

where the **`/usr/local/bin/qemu-aarch64-static`** binary is compiled for the native ABI of the host and hence is natively executed rather than wrapped by an image activator again.

Our work on the QEMU user mode[10] started with support for CHERI-RISC-V. Unfortunately, the upstream QEMU repository included an outdated BSD user mode implementation — initially developed for FreeBSD/mips64 in 2015[11, 12], also in collaboration with the University of Cambridge and SRI International. Thanks to the *qemu-bsd-user project*[13] improving the BSD user mode support in QEMU, we had a baseline CHERI-RISC-V user mode. We rebased these changes onto QEMU-CHERI, and extended the implementation. Main modifications included:

1. Improved system call interface implementation.
    a. System call arguments and results used integer types rather than data types corresponding to **`syscallarg_t`** from FreeBSD. We modified the system call interface implementation to use appropriate machine-independent data types, allowing us to handle CHERI capabilities.
    b. We changed QEMU to match closer the CheriBSD/FreeBSD system call interface implementation and generated a system call table for QEMU using the **`makesyscalls.lua`** script from CheriBSD derived from FreeBSD.
2. New data types **`abi_uintptr_t`** and **`abi_uintcap_t`**.
   We used the data types in places where integer data types were incorrectly used, not matching the actual machine-dependent data types storing pointers and capabilities.
3. CheriABI support, including ELF loading code, stack, **`mmap(2)`** implementations, and adapting existing system calls for CHERI capabilities.
4. Machine-dependent changes for CHERI-RISC-V and Arm Morello, including CHERI capability permission bits and capability register access routines.

This part of the project took us the longest time. Currently, the user mode itself can easily be used with the cheribuild *qemu-cheri-bsd-user* branch[2]. For example, you can run a CheriABI shell from a CheriBSD/riscv64c base system on a FreeBSD/amd64 host using:

```
$ ./cheribuild.py run-user-shell-riscv64-purecap
```

CheriBSD ports

Besides CHERI/CheriBSD-specific patches for software included in the FreeBSD ports collection, we introduced additional **`make(1)`** variables to allow modifying port build configurations depending on an ABI they are built for and allow building CheriABI packages with hybrid ABI compile-time dependencies:

- **`USE_PACKAGE_DEPENDS_REMOTE`**;
  When **`USE_PACKAGE_DEPENDS{, _ONLY}`** is enabled, try to install a package from a remote repository instead of building a port from scratch, if a local package does not exist.
- **`USE_PACKAGE_64_DEPENDS_ONLY`**;
  Install dependencies marked with **`USE_PKG64`** using their replacement hybrid ABI packages with **`pkg64`** instead of building them from scratch.
- **`USE_PKG64`**;
  When **`USE_PACKAGE_64_DEPENDS_ONLY`** is set, use a hybrid ABI package for a port that cannot be built for CheriABI and is required by another port that is being built for CheriABI.
- **`OPTIONS_{DEFINE,DEFAULT,EXCLUDE}_${ABI}`**;
  Lists of options specific to **`${ABI}`**.
- **`BROKEN_${ABI}`**.
  When set, a port is believed to be broken for **`${ABI}`**.

We also modified autoreconf, cmake, meson, ninja and python support to allow us to specify custom commands for hybrid ABI build utilities with `<UTILITY>_CMD make(1)` variables, e.g. `CMAKE_CMD`.

Poudriere

Our Poudriere fork[7] supports package building on both FreeBSD and CheriBSD hosts. By default, it uses base system tarballs for an operating system it is executed on but a user can specify the OS with a new flag `-o` for `poudriere-jail(8)`. Since CheriBSD does not include a toolchain in its base system, Poudriere installs it using `pkg` or `pkg64` within a Poudriere jail, outside a local base directory not to conflict with a toolchain built from CheriBSD ports. There are two set configurations shipped with Poudriere: `cheriabi` and `hybrida-bi`. Both use the same toolchain but define different `LOCALBASE` values, and the `cheriabi` one enables hybrid ABI build utilities that are unavailable for CheriABI.

Building CheriABI packages for the development branch on a CheriBSD/Morello host requires executing three simple commands:

```
$ poudriere jail -c -j aarch64c-dev -a arm64.aarch64c -v dev
$ poudriere ports -c -p main
$ poudriere bulk -j aarch64c-dev -p main -z cheriabi -a
```

When porting software to CHERI, CheriBSD users also can benefit from Poudriere to easily bootstrap a build environment. This is especially useful for hybrid ABI software that sometimes requires setting custom shared library search paths not to use by mistake CheriABI libraries from default search paths. With a Poudriere hybrid ABI jail, a developer does not have to worry about possible linking with CheriABI libraries as such jail only includes hybrid ABI programs and libraries.

Poudriere configuration and scripts

The last piece of the infrastructure is the *poudriere-infrastructure* repository[8] including Poudriere configuration files and shell scripts to bootstrap a build environment on a remote host, sign a package repository and deploy it at *pkg.CheriBSD.org*. In particular, `poudriere-remote.sh` builds the CheriBSD base system, the SDK, the QEMU user mode (if needed) and starts package building with Poudriere. Poudriere logs of CheriBSD package builds are publicly available at *poudriere.CheriBSD.org*.

Results

As of March 2023, CheriBSD provides 9104 CheriABI packages and 24494 hybrid ABI packages[4,5]. There are only 37 CheriBSD ports with patches. Most of the changes specific to CHERI have been successfully upstreamed to third-party software repositories. Some of the patches include changes specific to CHERI restrictions (e.g., the stronger pointer alignment to 16 bytes) which shows that open-source communities consider CHERI and Arm Morello as a promising platform.

CheriBSD users can easily set up a Morello host using a memstick installer obtained from CheriBSD.org (see Figure 2). `bsdinstall(8)` in CheriBSD includes an installer step[21] where a user can decide if they want to install packages to run a CheriABI graphical environment (using KDE Plasma and Wayland; see Figure 3) and additional hybrid ABI programs (at the moment Firefox and Chromium). These packages can easily be installed with meta-packages:

```
$ pkg64c install cheri-desktop
$ pkg64 install cheri-desktop-hybrid-extras
```

Figure 2: Arm Morello board running CheriBSD

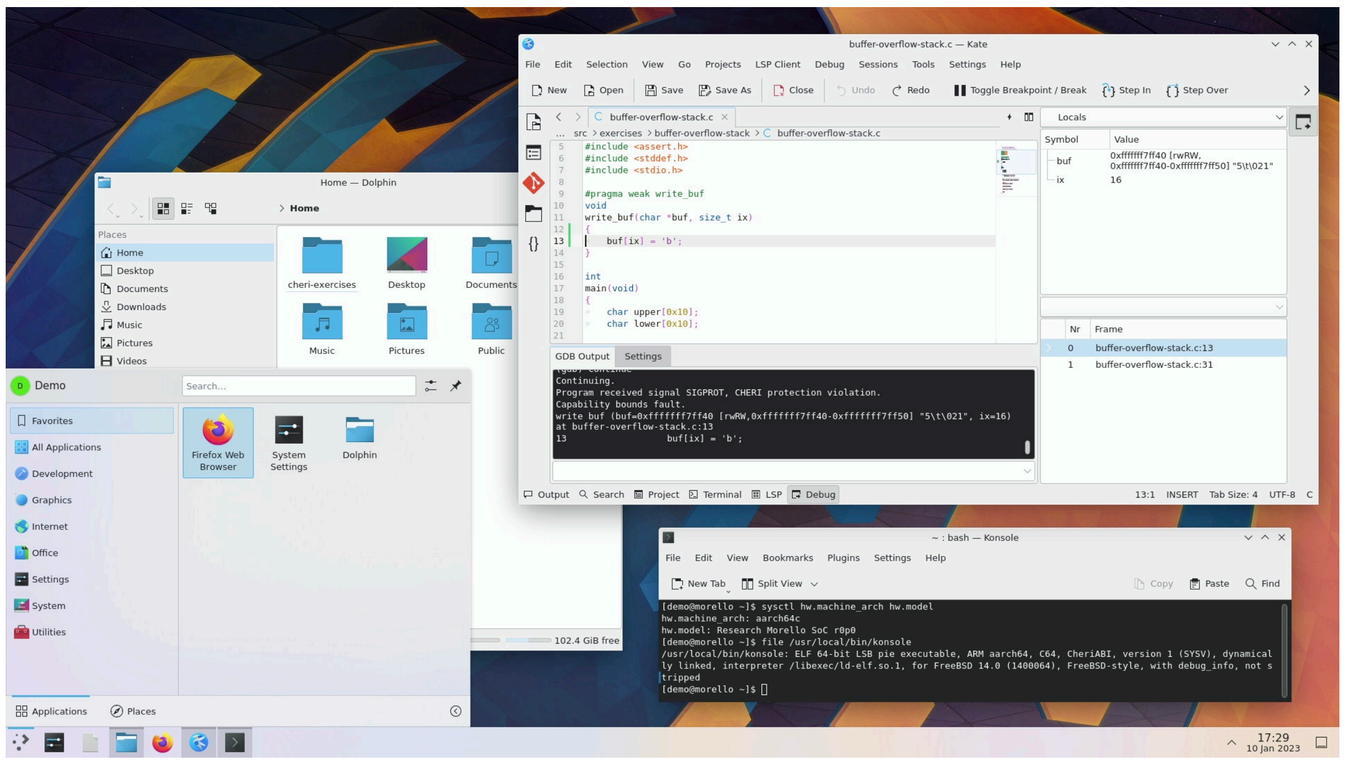

Figure 3: Memory-safe Morello desktop environment (CheriBSD, KDE Plasma, Wayland)[3]

CheriBSD releases and packages have been used by Technology Access Programme (TAP) participants. The Digital Security by Design (DSbD) initiative run by UK Research and Innovation organises TAP to allow UK-based companies experiment with the Arm Morello platform and prototype memory-safe projects. Currently, we collaborate with around 30 such companies. Thanks to the CheriBSD installer and pre-compiled third-party packages, TAP participants could easily deploy a work environment without the need to adapt for CheriABI, or even cross-compile, their software dependencies. However, many of them still needed to redesign their projects due to some missing third-party software, or port that software themselves.

Future work

Based on CheriBSD 22.05 and 22.12 releases, TAP, and CheriBSD users' experiences, we are planning next steps to increase the number of CHERI memory-safe packages. Currently, we are considering:

- CheriABI Python;
  As noted in the *Missing cross-ABI support* section, multiple build systems make use of

Python. Having Python adapted to CheriABI, we could not only build additional packages for CheriABI but also enable interesting research space in compartmentalising Python-based applications.

- Cross-ABI support in Poudriere;
  We would like to make use of hybrid ABI packages to build CheriABI packages with Poudriere. Currently, we build CheriABI packages using hybrid ABI build utilities by executing make package in port directories and transferring resulting packages to a package repository created with Poudriere. This feature would allow us to easily rebuild and deploy package repositories.
- Upstreaming patches;
  We would like to minimise the number of patches that must be maintained in CheriBSD ports, and instead commit them to upstream repositories. This includes changes in ports to better support custom local base paths in FreeBSD ports.
- CHERI-RISC-V packages.
  Currently, CheriBSD ships packages only for Arm Morello. Given that most of the applied patches are not specific to Morello, we should be able to build a large number of packages for CHERI-RISC-V as well. This would allow researchers to also evaluate the CHERI-RISC-V architecture against a large code corpus.

Conclusion

CheriBSD is a mature research operating system that can be used to prototype projects making use of new security primitives provided by CHERI, and as a development platform to test software against security vulnerabilities. While a lot of third-party software has been adapted for CHERI, there are still many crucial applications missing that would enable developing projects in new areas. We are excited to see the growing CHERI ecosystem, with at least 70 organisations from the Technology Access Programme[26], the CHERI within Defence and Security competition[27] and Digital Security by Design[28]. In the following months, we expect to continue our work to increase the number of available pure-capability packages.

References

1. cheribuild.py. https://github.com/CTSRD-CHERI/cheribuild
2. cheribuild.py. the qemu-cheri-bsd-user branch. https://github.com/CTSRD-CHERI/cheribuild/tree/qemu-cheri-bsd-user
3. CheriBSD. https://www.cheribsd.org/
4. CheriBSD packages. https://pkg.cheribsd.org/
5. CheriBSD Poudriere logs. https://poudriere.cheribsd.org/
6. CheriBSD ports. https://github.com/CTSRD-CHERI/cheribsd-ports
7. Poudriere extended for CheriBSD. https://github.com/CTSRD-CHERI/poudriere
8. Poudriere infrastructure for CheriBSD packages. https://github.com/CTSRD-CHERI/poudriere-infrastructure
9. QEMU with support for CHERI. https://github.com/CTSRD-CHERI/qemu
10. QEMU with support for CHERI. the qemu-cheri-bsd-user branch. https://github.com/CTSRD-CHERI/qemu/tree/qemu-cheri-bsd-user
11. BSDCan 2015: Embedded FreeBSD Development and Package Building via QEMU. https://www.bsdcan.org/2015/schedule/events/532.en.html
12. BSDCan 2015: Stacey Son. https://www.bsdcan.org/2015/schedule/speakers/267.en.html
13. The qemu-bsd-user project. https://github.com/qemu-bsd-user/qemu-bsd-user

14. The CHERI LLVM Compiler Infrastructure. https://github.com/CTSRD-CHERI/llvm-project
15. The GNU debugger extended to support CHERI. https://github.com/CTSRD-CHERI/gdb
16. Microsoft. CHERIoT: Rethinking security for low-cost embedded systems. https://www.microsoft.com/en-us/research/publication/cheriot-rethinking-security-for-low-cost-embedded-systems/
17. Arm. The CHERI LLVM Compiler Infrastructure. https://git.morello-project.org/morello/llvm-project
18. Morello Platform Software Repositories. https://git.morello-project.org/morello/docs
19. Arm. Morello Development Tools. https://developer.arm.com/Tools%20and%20Software/Morello%20Development%20Tools
20. Arm. Morello Program. https://www.arm.com/architecture/cpu/morello
21. Robert N. M. Watson, et al. Getting Started with CheriBSD. Installing on a Morello Board. https://ctsrd-cheri.github.io/cheribsd-getting-started/morello-install/
22. Robert N.M. Watson, et al. An Introduction to CHERI. Technical Report UCAM-CL- TR-941, University of Cambridge, Computer Laboratory, 2019.
23. Robert N. M. Watson, et al. Assessing the Viability of an Open-Source CHERI Desktop Software Ecosystem, Technical Report, Capabilities Limited, 17 September 2021.
24. Brooks Davis, et al. CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment. In Proceedings of 2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS'19). Providence, RI, USA, April 13-17, 2019.
25. Robert N. M. Watson, et al. Adversarial CHERI Exercises and Missions. https://ctsrd-cheri.github.io/cheri-exercises/
26. Digital Security by Design. Technology Access Programme Participants. https://www.dsbd.tech/whos-involved/technology-access-programme-participants/
27. Defence and Security Accelerator. Competition: CHERI within Defence and Security. https://www.gov.uk/government/publications/competition-cheri-within-defence-and-security
28. Digital Security by Design. Funded Projects. https://www.dsbd.tech/whos-involved/funded-projects/

KONRAD WITASZCZYK is a Research Associate and a PhD Student at the University of Cambridge working on the CHERI project. He graduated with a BSc degree in Theoretical Computer Science from the Jagiellonian University, an MSc degree in Computer Science from the University of Copenhagen and has been working with FreeBSD and its security-related mechanisms since 2013, including at Fudo Security. As part of his PhD, he is studying and working on compartmentalisation strategies for the CheriBSD kernel, and thus the FreeBSD kernel as well.