DSbD All Hands October 2022

Software porting to CHERI memory safety

Robert N. M. Watson, Simon W. Moore, Peter Sewell, Peter G. Neumann

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, Brooks Davis, 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, SRI International and Microsoft Research

Wolverhampton, 12 October 2022

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 contract FA8750-10-C-0237. 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.

What is the goal of this workshop?

Help and encourage people to port software to CHERI.

Scope of the workshop

The presentation and practice sessions cover:

CHERI C/C++ and CheriBSD/Morello as an example target environment

They do not cover:

Linux, Android environments and compartmentalisation models

Many ideas from these parts would still apply to Linux and Android.

We can discuss Linux, Android and compartmentalisation models during the discussion session.

Duplicate content

This presentation includes (but not only!) slides from previous events (CHERITech22, TAP webinars, …).

Some participants might be familiar with some content.

During the practical part, we can reorganise the room to split it into separate parts where participants can work on exercises or discuss their porting issues.

Mistakes ahead(?)

Some participants might disagree with some things I am going to say.

Feel free to interrupt and correct me during or after the workshop.

We plan to use these workshop materials in the future.

If you have any suggestions for improvements, please let me know.

Agenda

Start: 1:45 PM	
Part I: presentation **(~50 min)**	Work environment setup (VM in Azure with CheriBSD/Morello under QEMU)
	CHERI software stack (QEMU, LLVM, GDB, CheriBSD, ports, Poudriere)
	Porting issues at compile time (CHERI C/C++ compiler warnings)
	Porting issues at run time (CHERI C/C++ debugger support)
	Porting issues in CheriBSD ports (CheriABI, hybrid ABI)
Part II: practice **(~40 min)**	Example vulnerability
	Break: 15 min between 2:45 PM and 3:00 PM
	Example CHERI restrictions
Part III: discussion **(~30 min)**	Porting plans, suggestions and other topics
End: 4:00 PM	

Part I: presentation

CHERI software stack, multi-ABI support, porting issues.

Work environment setup: host

Host:

Virtual Machine running FreeBSD 13.1 in Azure.

Pre-installed software:

- tmux
- cheribuild and its dependencies
  - Runs CheriBSD/Morello in QEMU
  - Configured not to automatically update sources (in /cheri/cheribuild/cheribuild.json)
  - Configured to forward a host port 22000 to a guest port 22

Try to SSH into your host using your IP address and password:

ssh -l azureuser -L 127.0.0.1:22000:127.0.0.1:22000 <your-IP-address>

You can execute "**sudo su -l**" to become root.

Add "-o IdentityAgent=none" to ssh if you use ssh-agent with many keys (error: "Too many authentication failures").

Work environment setup: emulator

Start CheriBSD/Morello under QEMU on your host (takes ~3.5 min):

/cheri/cheribuild/cheribuild.py run-morello-purecap

You can use a serial console but we will use SSH.

Do not close a terminal with QEMU running as it would terminate CheriBSD/Morello.

Consider using tmux (pre-installed on your VM to prevent that).

Work environment setup: guest

Guest (slow!):

QEMU VM with CheriBSD/Morello 22.05p1; 1 vCPU, 2GB RAM, 180GB storage.

Pre-installed software and source code:

- Packages: llvm-base, gdb-cheri, tmux, sudo, vim, nano, and other but irrelevant here
- Poudriere
- Examples from the presentation part in ~/examples (check for updates: git -C ~/examples pull)
- Exercises from the practical part in ~/exercises (check for updates: git -C ~/exercises pull)
- CheriBSD ports in ~/cheribsd-ports

Try to SSH into your guest once you are connected to your host:

ssh -l cheriuser -p 22000 127.0.0.1

The password for cheriuser is "**cheriuser**".

You can execute "**sudo su -l**" to become root. The password for root is "**root**".

Add "-o IdentityAgent=none" to ssh if you use ssh-agent with many keys (error: "Too many authentication failures").

Work environment setup: questions?

SSH into your host using your IP address and password:

notebook % ssh -l azureuser -L 127.0.0.1:22000:127.0.0.1:22000 <your-Azure-VM-IP-address>

Start CheriBSD/Morello under QEMU on your host (takes ~3.5 min), preferably with tmux to prevent killing a session:

azure % /cheri/cheribuild/cheribuild.py run-morello-purecap

SSH into your guest once you are connected to your host:

notebook % ssh -l cheriuser -p 22000 127.0.0.1

The password is "**cheriuser**".

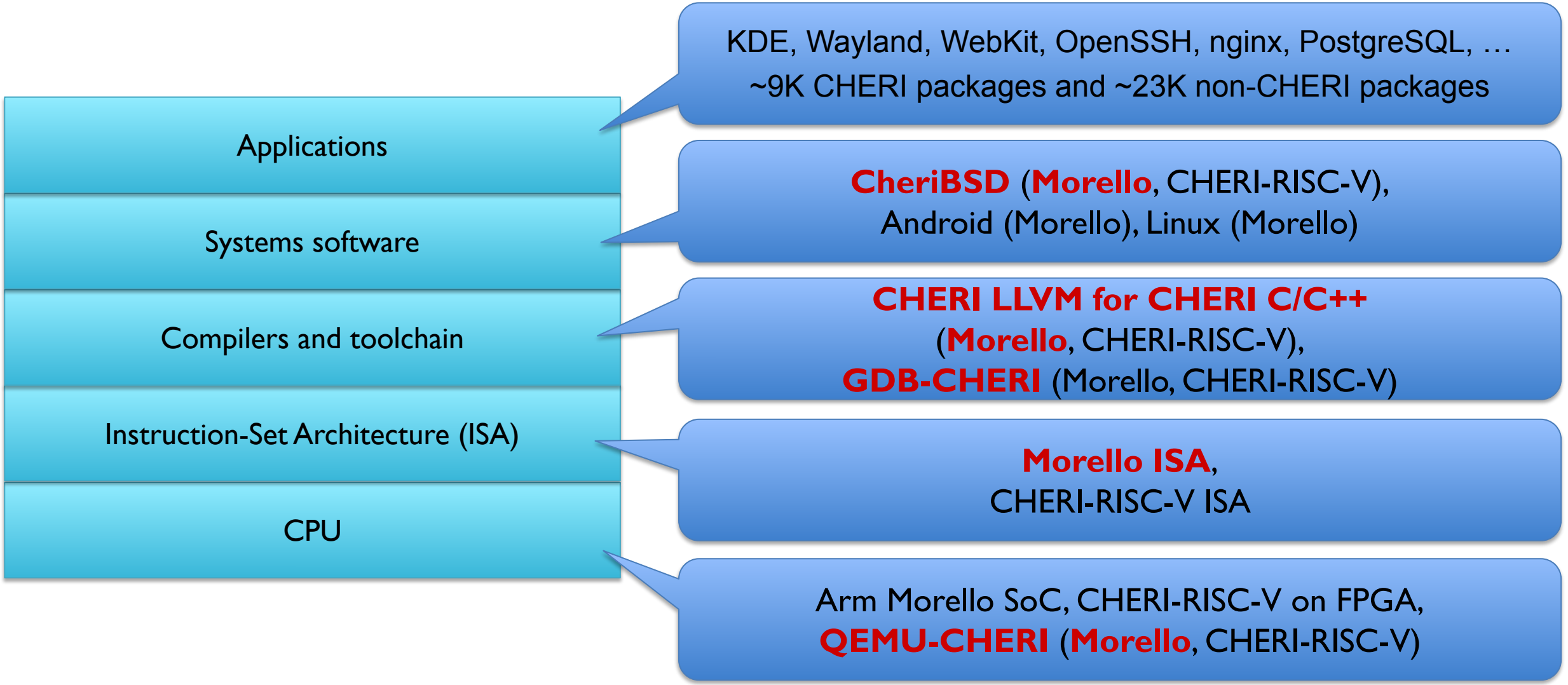
Update examples and exercises:

cheri % git -C ~/examples pull (https://github.com/kwitaszczyk/cheri-workshop-exercises)

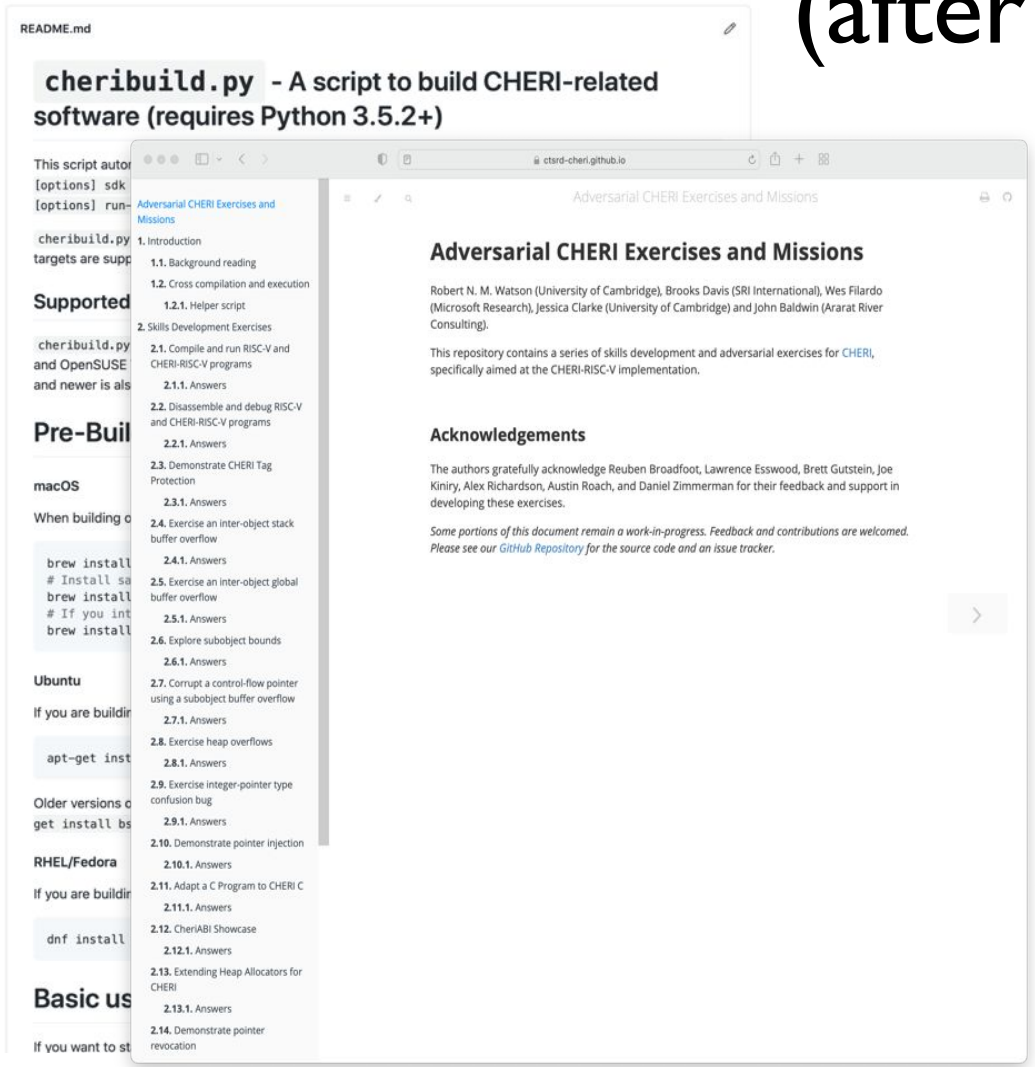
cheri % git -C ~/exercises pull (https://github.com/kwitaszczyk/cheri-workshop-examples)

Add "-o IdentityAgent=none" to ssh if you use ssh-agent with many keys (error: "Too many authentication failures").

CHERI (hardware-)software stack



How to experiment with CHERI (after this workshop)



Run an instance of CheriBSD/Morello in QEMU:

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

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

Try CHERI training exercises for developers, red teams, and bug bounties:

https://ctsrd-cheri.github.io/cheri-exercises/

Talk to the CHERI community:

https://ctsrd-cheri.github.io/cheribsd-getting-started/support/

https://bit.ly/3VcuZK5

cheribuild: fetch, build and run CHERI software

Run CheriBSD/Morello with the pure-capability world:

```
$ /cheri/cheribuild/cheribuild.py run-morello-purecap
$ /cheri/cheribuild/cheribuild.py run-morello-purecap --pretend
```

Get information on usage and available targets:

```
$ /cheri/cheribuild/cheribuild.py --help
$ /cheri/cheribuild/cheribuild.py --help-hidden |
    grep cheribsd-morello-purecap
```

cheribuild: JSON configuration support

Use JSON configuration files for multiple branches and scenarios:

$ ln -s /cheri/cheribuild/cheribuild.py /cheri/cheribuild/foo-cheribuild.py

$ /cheri/cheribuild/foo-cheribuild.py run-morello-purecap --pretend

cheribuild will try to use:

1. file.json specified with --config-file=file.json
2. (prefix-)cheribuild.json alongside (prefix-)cheribuild.py
3. ~/.config/(prefix-)cheribuild.json

We will use **/cheri/cheribuild/cheribuild.json** .

Cross-compiling is hard

Cross-compiling requires to:

- Use appropriate binutils with a compiler, a preprocessor, etc.
  - Code can assume binutils names or paths (e.g., cc)
- Set appropriate build system flags (e.g., --host and --build in configure)
- Set appropriate compiler flags to build for a target
  - Code can overwrite flags (e.g., LDFLAGS)
- Point a build system to a location with a sysroot
- Point a build system to a location with dependencies
  - Code can assume library paths (e.g., /usr/lib/libz.so)
- Install in a location that can be used by other software to look for dependencies

Utilities to cross-compile for CHERI

There are two main utilities that you can use to simplify cross-compiling for CHERI:

1. cheribuild can be easily extended with Python classes that specify build flags, compiler flags and location of dependencies.

2. QEMU user mode allows to emulate a native build environment in chroot or jail without the need to configure a build system to use custom toolchain and paths, and benefit from using native build utilities (e.g., FreeBSD ports).

cheribuild: cross-compiling

You can easily add new targets to cross-compile third-party software, e.g.

/cheri/cheribuild/pycheribuild/projects/cross/rsync.py

/cheri/cheribuild/pycheribuild/projects/cross/kde.py

Helper classes to easily use make, autotools, cmake, meson:

/cheri/cheribuild/pycheribuild/projects/cross/crosscompileproject.py

List available software to cross-compile:

$ /cheri/cheribuild/cheribuild.py --list-targets | grep purecap$

QEMU-CHERI: system mode vs user mode

There are two modes in upstream QEMU:

1. System mode allows to emulate a full operating system with processes running under it.

2. User mode allows to emulate processes without emulating a full operating system. CPU instructions are emulated but system calls are handled by a host kernel rather than an emulated kernel.

Both of these modes are implemented in QEMU-CHERI for Morello and CHERI-RISC-V.

QEMU-CHERI: system mode

- CHERI-RISC-V and Morello are supported;
- SMP does not work in practice: all vCPUs are emulated using a single host thread;
- Multi-threaded capability tag implementation exists for CHERI-RISC-V in a separate branch but might not be tested for Morello;
- We will not be able to use complex third-party software with dependencies but we can reproduce these examples on Morello hardware;
- Use cheribuild.py -p to find a QEMU command to customize it, e.g. change or add a disk.

QEMU-CHERI: BSD CheriABI user mode (1/2)

- CheriBSD/CHERI-RISC-V and CheriBSD/Morello are supported;
- Requires FreeBSD as a host not older than emulated CheriBSD;
- Developed to build software for CheriBSD and not run and evaluate software against CHERI;
- Tags are not validated due to CHERI tags and SoftMMU implementation. However, it can and should be fixed in the future;
- Can be used with Poudriere to build larger code scopes;
- We managed to build ~9K CheriABI packages this way.

QEMU-CHERI: BSD CheriABI user mode (2/2)

- Implemented in the qemu-cheri-bsd-user QEMU-CHERI branch;

- With the qemu-cheri-bsd-user cheribuild branch:

```
$ sudo ./cheribuild.py \
    --allow-running-as-root \
    --run-user-shell/jail \
    --run-user-shell/jail-extra-args mount.devfs \
    --run-user-shell/interpreter /libexec/ld-cheri-elf.so.1 \
    run-user-shell-riscv64-purecap
```

CHERI LLVM for Morello: packages

Users can install CHERI LLVM for Morello on CheriBSD with:

$ sudo pkg64 install llvm-base

llvm-base installs default toolchain binutils for the native ABI.

llvm-base requires llvm that installs default LLVM toolchain binutils.

llvm requires llvm-morello that installs CHERI LLVM for Morello.

CHERI LLVM for Morello: clang flags (1/3)

-march= specifies the target ARM architecture (ISA):

- morello
  - Morello with behaviour specified by **-mabi**.

-mabi= specifies the target ABI:

- aapcs (default);
  - Legacy AArch64 ABI with possible annotated pointers as capabilities;
  - ARM Architecture Procedure Call Standard.
- purecap.
  - Pure-capability ABI with all pointers as capabilities.

Explicitly set **-mabi** to your case as its default value might be changed to match the native ABI in the future.

CHERI LLVM for Morello: clang flags (2/3)

-Xclang -morello-vararg= specifies the varargs ABI:

- legacy (default on 22.05p1 but incompatible with 22.05p1)
- new (default in upstream and the next release)
  - For the hybrid ABI, pass varargs capabilities indirectly;
  - For CheriABI, pass varargs capabilities on the stack.

Explicitly set **-Xclang -morello-vararg=new** when using **clang**.
However, note that this flag might be removed in the future.

CHERI LLVM for Morello: clang flags (3/3)

ABI support is part of our research.

ABIs and compiler flags might change in the future, especially when it comes to compartmentalisation.

For example, compartmentalisation might require to:

- Indicate what run-time linker a binary should use
- Indicate what libraries should a binary be linked with to make use of compartmentalisation model.

CHERI LLVM for Morello: CC/CXX/CPP execution

For the native ABI (CheriABI in our case):

$ {cc,c++,cpp} -o helloworld-nativeabi helloworld.c

For CheriABI (the pure-capability ABI):

$ {clang,clang++,clang-cpp} -march=morello -mabi=purecap -Xclang -morello-vararg=new [...]

For the hybrid ABI:

$ {clang,clang++,clang-cpp} -march=morello -mabi=aapcs -Xclang -morello-vararg=new [...]

CHERI LLVM for Morello: printing addresses/capabilities

Format	Meaning	CheriABI: void *	Hybrid ABI: void *	Hybrid ABI: void * __capability
%p	Expect a pointer. Print an address.	Prints an address.	Prints an address.	Emits a warning: expects a pointer. Prints an address.
%#p	Expect a pointer. Print a pointer.	Prints a capability.	Prints an address.	Emits a warning: expects a pointer. Prints an address.
%lp	Expect a capability. Print an address.	Prints an address.	Emits a warning: expects a capability. Disallowed.	Prints an address.
%#lp	Expect a capability. Print a pointer.	Prints a capability.	Emits a warning: expects a capability. Disallowed.	Prints a capability.

https://github.com/CTSRD-CHERI/cheri-c-programming/wiki/Displaying-Capabilities

GDB-CHERI: disassembler

GDB-CHERI 8.3 cannot disassemble capability instructions:

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x00000000000110acc <+0>:     .inst   0x028203ff ; undefined
```

GDB-CHERI 12 can disassemble capability instructions and will be delivered as a package for CheriBSD 22.10.

CHERI LLVM for Morello: llvm-mc

Disassemble machine code:

```
$ echo "0xff 0x03 0x82 0x02" |
    llvm-mc -mattr=+morello --disassemble
```

Construct an object file:

```
$ echo ".4byte 0x028203ff" |
    llvm-mc -mattr=+morello -filetype=obj |
    hexdump -C
```

CHERI LLVM for Morello: llvm-objdump

Disassemble a file:

```
$ objdump -D -j .text /libexec/ld-elf.so.1
$ llvm-objdump -D -j .text --mattr=+morello /libexec/ld-elf.so.1
```

Disassemble machine code from stdin:

```
$ echo ".4byte 0x028203ff" |
    llvm-mc -mattr=+morello -filetype=obj |
    llvm-objdump -D -j .text --mattr=+morello -
```

GDB-CHERI: shell disassemble

```
(gdb) disassemble
Dump of assembler code for function main:
=> 0x00000000000110acc <+0>:     .inst   0x028203ff ; undefined
(...)

(gdb) x/4wx $pcc

(gdb) shell ~/examples/disassemble word1 word2 word3 word4
```

Author: Jessica Clarke, University of Cambridge

GDB-CHERI: capability information

GDB can decode capabilities and print their properties:

```
(gdb) info register x1 c1
x1          0xffffff7feb8      281474976186040
c1          0xdc5d40007ec0feb80000ffffff7feb8 0xffffff7feb8 [rwRW,0xffffff7feb8-0xffffff7fec0]
            <address> [<permissions>,<base>-<top>] (<attr>)
            top is upper bound plus 1
```

https://github.com/CTSRD-CHERI/cheri-c-programming/wiki/Displaying-Capabilities

GDB-CHERI: tag information

GDB-CHERI 8.3 does not print if a tag is valid for capability registers and in-memory capabilities.

For capability registers, you can use the (fake) tag_map register:

(gdb) p/x $tag_map & (1 << 2)

For in-memory capabilities, use debug symbols and printf(3) (%#p).

GDB-CHERI 12 can print tag information for capability registers and in-memory capabilities.

Porting issues at compile time: loss of provenance

compiler-warnings.c:3:3: warning: cast from provenance-free integer type to pointer type will give pointer that can not be dereferenced [-Wcheri-capability-misuse]

```
*(char *)x = 'A';
 ^
```

A developer must use an appropriate data type to indicate its intentions: what value is stored in a variable. In this case: **x** should use a data type indicating it can store a pointer.

Porting issues at compile time: ambiguous provenance

compiler-warnings.c:13:44: warning: binary expression on capability types 'unsigned __intcap' and 'uintptr_t' (aka 'unsigned __intcap');

it is not clear which should be used as the source of provenance; currently provenance is inherited from the left-hand side [-Wcheri-provenance]

```
newptr = (void *)(((uintptr_t)ptr1 & 0x3) | (uintptr_t)ptr2);
                  ~~~~~~~~~~~~~~~~~~~~~~~ ^ ~~~~~~~~~~~~~~~
```

CHERI capabilities can be created using exactly one other capability.
A developer must indicate in a complex expression which capability is a source capability to construct a new one.

Porting issues at compile time: underaligned capability

```
compiler-warnings.c:21:4: warning: alignment (8) of 'struct (unnamed struct at
compiler-warnings.c:19:2)' is less than the required capability alignment (16)
[-Wcheri-capability-misuse]
        } obj __attribute__((aligned(8)));
```

Each in-memory capability has an associated tag that corresponds to a capability-aligned address.
A developer must align data so that capabilities within them are aligned to a capability size.

Porting issues at run time: misaligned copying

Use a temporary Parse object to copy it

Browse files

```
In order to correctly copy capabilities from the recursive region of
a Parse object, we must make sure that the copied part and a destination
buffer it is copied to have the same addresses modulo the capability
alignment.

Instead of appropriately aligning the destination buffer, let's use
a Parse object to make sure that any future changes in the Parse
structure would not introduce capability misalignments when copying the
recursive region.

Another solution to this problem would be to move the recursive region
to a separate structure but that would introduce additional changes
wherever Parse fields from the recursive regions are accessed.
```

version-3.39.0-cheriabi

kwitaszczyk committed on 2 Sep Verified 1 parent 5402648 commit 02fc6085e57f52f08148c985ce846b2490607dc8

Showing **1 changed file** with **3 additions** and **3 deletions**. Split Unified

6 src/build.c

```
@@ -311,7 +311,7 @@ void sqlite3NestedParse(Parse *pParse, const char *zFormat, ...){
311 311   char *zSql;
312 312   sqlite3 *db = pParse->db;
313 313   u32 savedDbFlags = db->mDbFlags;
314     -   char saveBuf[PARSE_TAIL_SZ];
    314 +   Parse saveParse;
315 315
316 316   if( pParse->nErr ) return;
317 317   assert( pParse->nested<10 );  /* Nesting should only be of limited depth */
@@ -327,15 +327,15 @@ void sqlite3NestedParse(Parse *pParse, const char *zFormat, ...){
327 327     return;
328 328   }
329 329   pParse->nested++;
330     -   memcpy(saveBuf, PARSE_TAIL(pParse), PARSE_TAIL_SZ);
    330 +   memcpy(PARSE_TAIL(&saveParse), PARSE_TAIL(pParse), PARSE_TAIL_SZ);
331 331   memset(PARSE_TAIL(pParse), 0, PARSE_TAIL_SZ);
332 332   db->mDbFlags |= DBFLAG_PreferBuiltin;
333 333   sqlite3RunParser(pParse, zSql);
334 334   sqlite3DbFree(db, pParse->zErrMsg);
335 335   pParse->zErrMsg = 0;
336 336   db->mDbFlags = savedDbFlags;
337 337   sqlite3DbFree(db, zSql);
338     -   memcpy(PARSE_TAIL(pParse), saveBuf, PARSE_TAIL_SZ);
    338 +   memcpy(PARSE_TAIL(pParse), PARSE_TAIL(&saveParse), PARSE_TAIL_SZ);
339 339   pParse->nested--;
340 340  }
341 341
```

https://github.com/CTSRD-CHERI/sqlite/commit/02fc6085e57f52f08148c985ce846b2490607dc8

https://bit.ly/3VcuZK5

Porting issues at run time: partial capability copying

Copy first items with memcpy(). Browse files

```
Perl_repeatcpy() copies a number of items of the same length specified
by count and len, respectively, in three runs:
1. PERL_REPEATCPY_LINEAR (4) items byte by byte.
2. 2^N items with a single memcpy() call, for N in [4; count/2].
3. Remaining items with a single memcpy() call.

In case a capability is copied with Perl_repeatcpy(), the first run
(byte by byte) stores untagged capabilities while the rest works as
expected.

Replace the first run with memcpy() calls to store valid (tagged)
capabilities.
```

v5.32.1-cheriabi

kwitaszczyk committed on 3 Aug Verified 1 parent 05dade1 commit 84d6100a67c1775e73865990cddb00bc7e4975d6

Showing **1 changed file** with **2 additions** and **4 deletions**. Split Unified

6 util.c

```
@@ -3096,10 +3096,8 @@ Perl_repeatcpy(char *to, const char *from, I32 len, IVINT count)
3096 3096
3097 3097           linear = count < PERL_REPEATCPY_LINEAR ? count : PERL_REPEATCPY_LINEAR;
3098 3098           for (items = 0; items < linear; ++items) {
3099      -            const char *q = from;
3100      -            IV todo;
3101      -            for (todo = len; todo > 0; todo--)
3102      -                *p++ = *q++;
     3099 +            memcpy(p, from, len);
     3100 +            p += len;
3103 3101           }
3104 3102
3105 3103           half = count / 2;
```

https://github.com/CTSRD-CHERI/perl5/commit/84d6100a67c1775e73865990cddb00bc7e4975d6

https://bit.ly/3VcuZK5

Porting issues at run time: assumed pointer size

Correctly fill mode with zeroes.

Browse files

Use the actual size of the mode array - PERL_MODE_MAX instead of a size of the mode pointer.

v5.32.1-cheriabi

kwitaszczyk committed on 2 Apr Verified 1 parent e123955 commit f964f8e74e0e0d4c82087da9c139e0c6f44869df

Showing **1 changed file** with **1 addition** and **1 deletion**. Split Unified

2 doio.c

```
@@ -353,27 +353,27 @@
353 353     DO_PIPEOPEN_EXPERIMENTING_CLOEXEC(PL_strategy_socketpair, pairfd,
354 354         PerlSock_socketpair(domain, type | SOCK_CLOEXEC, protocol, pairfd),
355 355         PerlSock_socketpair(domain, type, protocol, pairfd));
356 356 #  else
357 357     DO_PIPEOPEN_THEN_CLOEXEC(pairfd,
358 358         PerlSock_socketpair(domain, type, protocol, pairfd));
359 359 #  endif
360 360 }
361 361 #endif
362 362
363 363 static IO *
364 364 S_openn_setup(pTHX_ GV *gv, char *mode, PerlIO **saveifp, PerlIO **saveofp,
365 365               int *savefd,  char *savetype)
366 366 {
367 367     IO * const io = GvIOn(gv);
368 368
369 369     PERL_ARGS_ASSERT_OPENN_SETUP;
370 370
371 371     *saveifp = NULL;
372 372     *saveofp = NULL;
373 373     *savefd = -1;
374 374     *savetype = IoTYPE_CLOSED;
375 375
376     -   Zero(mode,sizeof(mode),char);
    376 +   Zero(mode, PERL_MODE_MAX, char);
377 377     PL_forkprocess = 1;         /* assume true if no fork */
378 378
379 379     /* If currently open - close before we re-open */
```

https://github.com/CTSRD-CHERI/perl5/commit/f964f8e74e0e0d4c82087da9c139e0c6f44869df

https://bit.ly/3VcuZK5

Porting issues at run time: incorrect {,u}intptr_t definition

print/texinfo: don't redefine intptr_t for CheriABI Browse files

When compiling for CheriABI, don't redefine {,u}intptr_t as integer data types.

main

kwitaszczyk committed 5 days ago Verified 1 parent 15daad7 commit 1c740aecda79044d5cbcafca15fce0f586fc66da

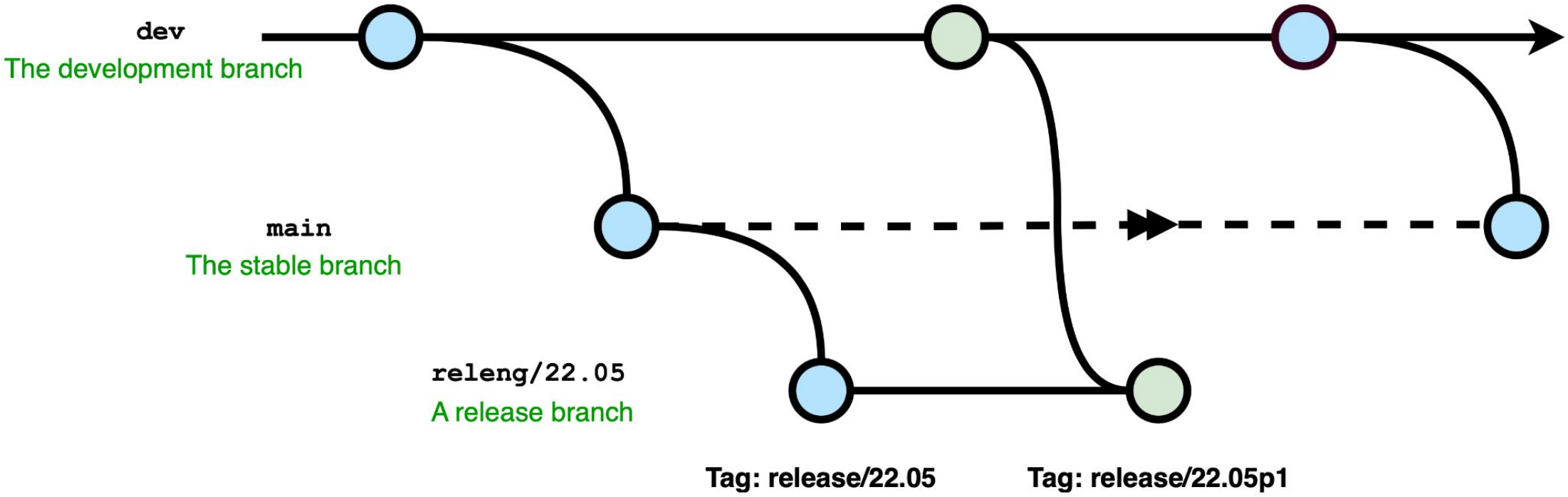
Showing **1 changed file** with **11 additions** and **0 deletions.** Split Unified

11 print/texinfo/files/cheribsd.patch

```
@@ -0,0 +1,11 @@
+--- tp/Texinfo/XS/gnulib/lib/stdint.in.h.orig   2022-10-06 15:01:42 UTC
+++ tp/Texinfo/XS/gnulib/lib/stdint.in.h
@@ -306,7 +306,7 @@ typedef gl_uint_fast32_t gl_uint_fast16_t;
     uintptr_t to avoid conflicting declarations of system functions like
     _findclose in <io.h>.  */
  # if !((defined __KLIBC__ && defined _INTPTR_T_DECLARED) \
-       || defined __MINGW32__)
+       || defined __MINGW32__ || defined __CHERI_PURE_CAPABILITY__)
  #  undef intptr_t
  #  undef uintptr_t
  #  ifdef _WIN64
```

https://github.com/CTSRD-CHERI/cheribsd-ports/commit/1c740aecda79044d5cbcafca15fce0f586fc66da

CheriBSD: release engineering



Graph template from: https://www.bryanbraun.com/2020/04/24/drawing-git-branching-diagrams/

CheriBSD: ABI version

CheriBSD ABI version is defined in /usr/include/sys/param.h:

```
$ grep ‘define __CheriBSD_version’ /usr/include/sys/param.h
```

Check a CheriBSD ABI version of a binary with:

```
$ readelf -n /bin/sh
```

Package managers use the ABI version to select a package repository:

```
$ (pkg64c -vv; pkg64 -vv) | grep ^ABI
$ grep ABI /etc/pkg/CheriBSD.conf /etc/pkg64/CheriBSD.conf
```

CheriBSD: multi-ABI support

Value \ ABI	CheriABI	Hybrid ABI
LOCALBASE / PREFIX	/usr/local	/usr/local64
CC / CXX / CPP	cc / c++ / cpp	clang / clang++ / clang-cpp
Run-time linker	/libexec/ld-elf.so.1	/libexec/ld-elf64.so.1
Default library directories	/lib:/usr/lib:/usr/local/lib	/usr/lib64:/usr/local64/lib
Shared library search directories	LD_LIBRARY_PATH	LD_64_LIBRARY_PATH
Package manager	pkg64c	pkg64

CheriBSD ports: CheriABI software

Users can build third-party software (with their patches) using CheriBSD ports.

You can use packages for dependencies if you do not want to build them yourself:

```
$ git clone https://github.com/CTSRD-CHERI/cheribsd-ports.git

$ cd cheribsd-ports/my/port

$ sudo make install USE_PACKAGE_DEPENDS=1 USE_PACKAGE_DEPENDS_REMOTE=1

$ sudo echo USE_PACKAGE_DEPENDS=1 >>/etc/make.conf
$ sudo echo USE_PACKAGE_DEPENDS_REMOTE=1 >>/etc/make.conf
```

Have a look at Mk/bsd.port.mk in CheriBSD ports for more useful flags/variables.

CheriBSD ports: hybrid ABI software

It is possible to build CheriBSD ports for the hybrid ABI.
You would have to set:

- ARCH=aarch64
- MACHINE_ARCH=aarch64
- LOCALBASE=/usr/local64
- CC=/usr/local64/bin/clang

and other variables when executing make.

FreeBSD ports and CheriBSD ports do not have any notion of an ABI.

You will find many ports broken when compiling for the hybrid ABI while running CheriABI world.

Poudriere: installation

Fetch our Poudriere fork:
$ git clone https://github.com/CTSRD-CHERI/poudriere

Install Poudriere:
$ cd poudriere; ./configure; make; sudo make install
$ sudo cp src/etc/poudriere.d/*make.conf.sample /usr/local/etc/poudriere.d/
$ sudo cp -a src/share/poudriere/toolchain /usr/local/share/poudriere/toolchain

Set FREEBSD_HOST to download.CheriBSD.org in: /usr/local/etc/poudriere.conf .

Install make.conf files:
$ sudo cp /usr/local/etc/poudriere.d/make.conf.sample /usr/local/etc/poudriere.d/make.conf
$ sudo cp /usr/local/etc/poudriere.d/cheriabi-make.conf.sample /usr/local/etc/poudriere.d/cheriabi-make.conf
$ sudo cp /usr/local/etc/poudriere.d/hybridabi-make.conf.sample /usr/local/etc/poudriere.d/hybridabi-make.conf

Poudriere: build hybrid ABI software

Create an aarch64 (hybrid ABI for Morello) jail with CheriBSD 22.05p1:

```
$ sudo poudriere jail -c -j aarch64-22_05p1 -a arm64.aarch64 -v 22.05p1
```

Register a CheriBSD ports directory as a ports tree:

```
$ sudo poudriere ports -c -p main -m null -M /usr/home/cheriuser/cheribsd-ports
```

Start the aarch64 jail with the CheriBSD ports directory mounted and make flags for the hybrid ABI:

```
$ sudo poudriere jail -s -j aarch64-22_05p1 -p main -z hybridabi
```

Find a jail without an IP address and use it to build ports mounted in /usr/ports:

```
$ jls
$ sudo jexec <jid>
```

Poudriere: build CheriABI software

Create an aarch64c (CheriABI for Morello) jail:

$ sudo poudriere jail -c -j aarch64c-22_05p1 -a arm64.aarch64c -v 22.05p1

CheriABI and hybrid ABI builds can use the same ports tree.

Start the aarch64c jail with the CheriBSD ports directory mounted and make flags for CheriABI:

$ sudo poudriere jail -s -j aarch64-22_05p1 -p main -z cheriabi

Find a jail without an IP address and use it to build ports mounted in /usr/ports:

$ jls

$ sudo jexec <jid>

Porting issues in CheriBSD ports

Affecting CheriABI builds:

- DOCS
- Missing Python
  - devel/autoconf, devel/meson, …
- #undef uintptr_t
  #define uintptr_t long int
  - gnulib, e.g. in print/texinfo
- Major port upgrades
  - net/rsync

Affecting hybrid ABI builds:

- Missing -L${LOCALBASE}
- Hardcoded LOCALBASE / PATH in a port;
  - www/chromium
- Hardcoded LD_LIBRARY_PATH environment variable name;
  - lang/python39
- Hardcoded CC / CXX / CPP paths;
- Hardcoded library paths.
  - /usr/lib/libz.so instead of /usr/lib64/libz.so

CheriBSD ports: CheriBSD patches

CheriBSD ports must be patched to:

- Compile for CheriABI
- Build with non-default values (e.g., LOCALBASE)
- Behave differently on CheriBSD (e.g., use pkg64 for the hybrid ABI)

Most patches are placed in files/cheribsd.patch in port directories.

cheribsd.patch is applied after all other patches from FreeBSD.

The goal is to minimise the number of CheriBSD-patches and instead upstream all changes to port-specific repositories and FreeBSD ports.

Part II: practice

Example vulnerabilities, CHERI restrictions.

Exercise: vulnerability

1. An example buffer overflow is in ~/exercises/.
2. Compile a program for the hybrid ABI.
3. Compile a program for CheriABI.
   Consider adding -g for debug symbols.
4. Run programs for both ABIs.
5. For the CheriABI program:
   a. Enter gdb with a core dump.
   b. Analyse with gdb what instruction failed and why.
   c. Read the code and try to fix the issue.

Exercise: CHERI restrictions (1/2)

1. A broken cat sits in ~/exercises/.
2. Compile a program for the hybrid ABI.
3. Compile a program for CheriABI.
   Ignore compiler output for now.
4. Run for both ABIs: ./cat /etc/hostid
5. For the CheriABI program:
   a. Read the code and find the place where the program exits.
   b. Analyse with gdb why it fails.
      Consider setting a breakpoint.
   c. Look at compiler output and try to fix the issue.

Exercise: CHERI restrictions (2/2)

6. Run for both ABIs: ./cat -n /etc/hostid
7. For the CheriABI program:
   a. Enter gdb with a core dump.
   b. Analyse with gdb what instruction failed and why.
   c. Look at the compiler output and try to fix the issue.

Part III: discussion

Porting plans, suggestions and other topics.

Discussion

1. What software would you like to see ported to CHERI, e.g. a missing dependency for your project?
2. Are you currently porting any software to CheriBSD/Linux/Android CHERI/CHERI-RISC-V/Morello?
3. Would you like to see your ported software in CheriBSD ports?
4. How to organise software porting to CHERI so that it can be used across different targets, in Linux/Morello, CheriBSD/Morello, CheriBSD/CHERI-RISC-V?

Place for notes: https://bit.ly/3eclutT

R&D: CheriBSD features in fall 2022

Feature	Status
GUI with GPU support	• Hybrid and pure-capability Morello GPU kernel drivers; • Hybrid ABI and CheriABI compilation of user space; • Increasing number of ports adapted to CheriABI (Qt 5, KDE, Wayland).
GDB 12	• Disassembler for capability instructions; • Register and in-memory capability tag validity information.
ZFS	• Experimental implementation; • Allows to create a zpool for a single drive; • Multiple disks and boot environments do not work yet.
3rd-party packages	• Upstream FreeBSD ports are merged every two weeks now; • ~9K CHERI packages, ~23K non-CHERI packages, and expected to increase (e.g., with the GUI stack).
Linker-based compartmentalization	• Prototype runs some UNIX applications; • Should be included as an optional run-time linker in the next release.

R&D: other projects in progress

Feature	Status
Co-process compartmentalization	• Prototype runs some compartmentalized software (e.g., OpenSSL); • API co-design.
Userlevel heap temporal safety	• Prototype runs SPEC benchmarks; • Should be available as snapshots at download.CheriBSD.org; • Stub syscalls and libc symbols should be included in the next release.
bhyve (Type-2) hypervisor	• Prototype boots pure-capability guest OS; • More testing and review required.

CHERI ecosystem community

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

https://cheri-cpu.org/

(→ CHERI → CHERI-CPU Slack)

The Slack includes the channel #workshop-dsbd-22-10.

Thanks for your attention!

You can contact me at:

konrad.witaszczyk@cl.cam.ac.uk

Feedback welcome!