Porting C/C++ software to Morello

Alex Richardson

CHERI C/C++

- Pure-capability environments use a C/C++ variant we call CHERI C/C++.
- CHERI C/C++ is very similar to “normal” C/C++ with a few difference such as:
  - On Morello, pointers require 16-byte alignment.
  - `(u)intptr_t` is not the same type as `(unsigned) long`.
  - Pointers created from a (non-`uintptr_t`) integer are not dereferenceable.
  - Pointers are tightly bounded and cannot be used to access adjacent objects.
- With C99 (or better C11) features such as `(u)intptr_t` and `max_align_t`, targeting CHERI C/C++ is mostly a matter of using correct types.

CHERI C/C++

- CHERI C/C++ is **highly compatible with existing code**.
  - In many cases, no changes are required to run code on Morello!
  - This is especially likely for higher-level C++ code (e.g. desktop applications) – for KDE on X11 it was only **0.026%** of the 6M SLoC.
- However, some (low-level) code uses patterns that are not compatible with the strict provenance semantics enforced by CHERI C/C++
- In general, only language runtimes or OS kernels might require significant adaptation.
- For other projects, minor changes such as changing individual casts or increasing the alignment of custom allocators should be sufficient.

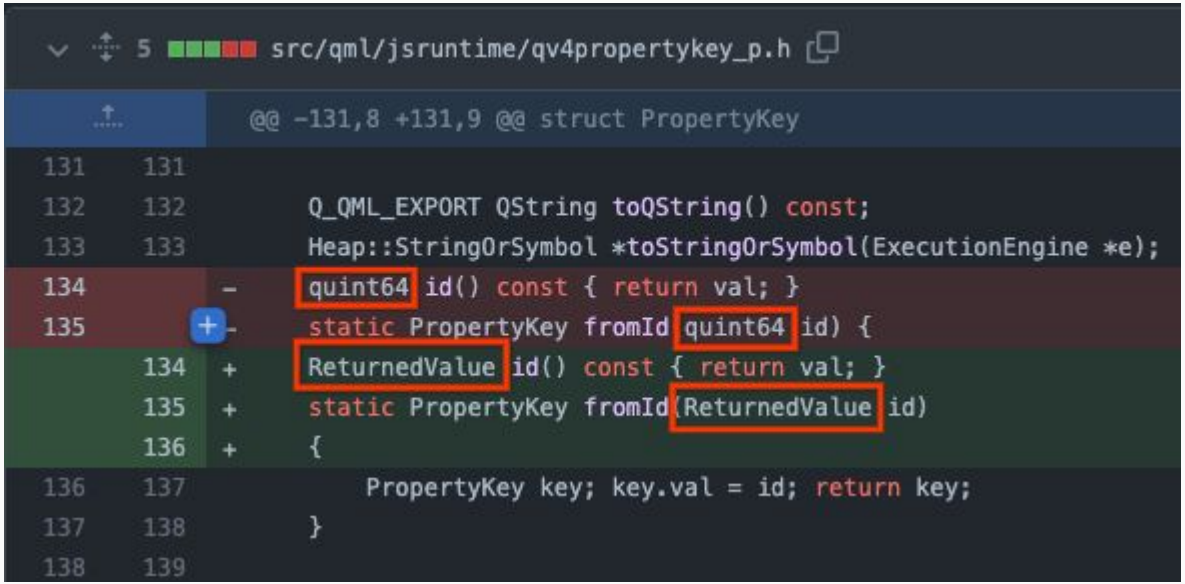
Converting integers to pointers

- In CHERI C/C++ `unsigned long` cannot store the capability metadata
  - Casting from pointer to integer strips the capability metadata.
  - Usually flagged by the compiler by emitting a warning when creating a pointer from an integer.
- Casting via uintptr_t generally resolves this problem.

```
  #ifdef WITH_SIMD
-   cvalue = values = (JCOEF *)PAD((size_t)values_unaligned, 16);
+   cvalue = values = (JCOEF *)PAD((JUINTPTR)values_unaligned, 16);
  #else
    /* Not using SIMD, so alignment is not needed */
    cvalue = values = values_unaligned;

  @@ -945,7 +946,7 @@ encode_mcu_AC_refine(j_compress_ptr cinfo, JBLOCKROW *M

        emit_restart(entropy, entropy->next_restart_num);

  #ifdef WITH_SIMD
-   cabsvalue = absvalues = (JCOEF *)PAD((size_t)absvalues_unaligned, 16);
+   cabsvalue = absvalues = (JCOEF *)PAD((JUINTPTR)absvalues_unaligned, 16);
```

Converting integers to pointers: -Wshorten-cap-to-int

- Truncating capability metadata can result in crashes if converted back to a pointer.
- Based on 32 → 64-bit transition warning `-Wshorten-64-to-32`
- Real-world [example from QtDeclarative] (explicit uint64_t for 32-bit systems):



Checking for 64-bit architectures (__LP64__)

- The __LP64__ macro is often used to detect whether registers are 64 or 32 bits.
- This macro **is not defined** for CHERI C/C++, which can cause software to assume a 32-bit architecture (example fix [from the X11 libraries]):

include/X11/Xmd.h +5 -1 Viewed

```
@@ -57,7 +57,11 @@ SOFTWARE.
57  57   #  include <sys/isa_defs.h> /* Solaris: defines _LP64 if necessary */
58  58   # endif
59  59
60      - # if defined (_LP64) || defined(__LP64__) || \
    60  + #if defined(__SIZEOF_LONG__)
    61  + # if __SIZEOF_LONG__ == 8
    62  + #  define LONG64                                /* 32/64-bit architecture */
    63  + # endif
    64  + # elif defined (_LP64) || defined(__LP64__) || \
61  65         defined(__alpha) || defined(__alpha__) || \
62  66         defined(__ia64__) || defined(ia64) || \
63  67         defined(__sparc64__) || \
```

Ambiguous provenance

- CHERI C/C++ using a single-provenance semantics, i.e. every pointer must be derived from exactly one other pointer.
- For binary arithmetic operations on `(u)intptr_t`, the compiler might not be able to determine which operand is a pointer and which one is the offset/mask.
- The fallback behaviour is to use the left-hand operand (which is usually correct).
- Unlike addition, subtraction and bitwise-& can return either a pointer or an integer.
- Fixed by casting to a non-provenance-carrying type (example from FontConfig).

```
   /* Compute pointer offset */
-  #define FcPtrToOffset(b,p)      ((intptr_t) (p) - (intptr_t) (b))
+  #define FcPtrToOffset(b,p)      ((ptrdiff_t) ((intptr_t) (p) - (intptr_t) (b)))

   /* Given base address, offset and type, return a pointer */
-  #define FcOffsetToPtr(b,o,t)    ((t *) ((intptr_t) (b) + (o)))
+  #define FcOffsetToPtr(b,o,t)    ((t *) ((intptr_t) (b) + (ptrdiff_t) (o)))

   /* Given base address, encoded offset and type, return a pointer */
   #define FcEncodedOffsetToPtr(b,p,t) FcOffsetToPtr(b,FcOffsetDecode(p),t)
```

When the compiler can’t help anymore

- Running programs under gdb (`gdb -ex=r --args <cmd>`) will generally stop close to where the underlying issue is.
- In many cases, the incorrect arithmetic, etc. will only be one or two stack frames up

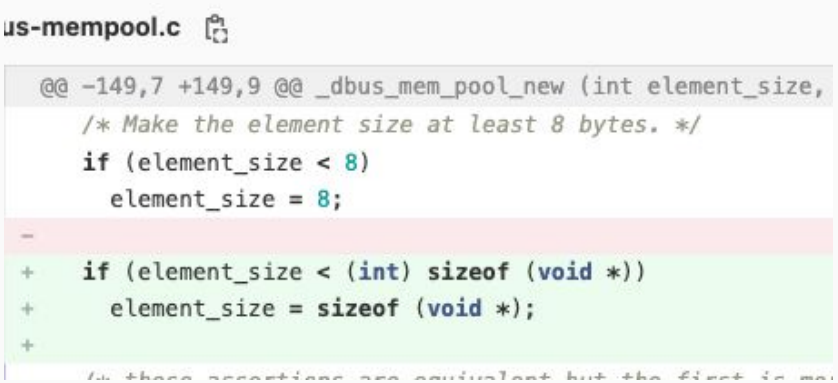
```
Starting program: /opt/cheri-exercises/buffer-overflow-stack-cheri

Program received signal SIGPROT, CHERI protection violation
Capability bounds fault caused by register ca1.
0x0000000000101dae in write_buf (buf=0x3fffdfff5c [rwRW,0x3fffdfff5c-0x3fffdfff6c] "", ix=16)
at src/exercises/buffer-overflow-stack/buffer-overflow-stack.c:13
13      src/exercises/buffer-overflow-stack/buffer-overflow-stack.c: No such file or
directory.
(gdb) bt
#0  0x0000000000101dae in write_buf (buf=0x3fffdfff5c [rwRW,0x3fffdfff5c-0x3fffdfff6c] "",
ix=16) at src/exercises/buffer-overflow-stack/buffer-overflow-stack.c:13
#1  0x0000000000101e98 in main () at src/exercises/buffer-overflow-stack/buffer-overflow-
stack.c:31
(gdb)
```

Incorrectly aligned capabilities

- Some projects have custom allocators (or wrap malloc to insert additional metadata). However, in many cases these allocators **hardcode 8 byte alignment**.
- If C11 can be used, aligning to `_Alignof(max_align_t)` is the correct fix, and if not a patch with a type that matches the intended usage may be upstreamable.

```
@@ -139,8 +139,9 @@ static void *sqlite3MemMalloc
  sqlite3_int64 *p;
  assert( nByte>0 );
  testcase( ROUND8(nByte)!=nByte );
-  p = SQLITE_MALLOC( nByte+8 );
+  p = SQLITE_MALLOC( nByte+sizeof(uintptr_t) );
  if( p ){
+    p += (sizeof(uintptr_t)/sizeof(int64_t))-1;
    p[0] = nByte;
```

[Wrapped malloc() in SQLite]

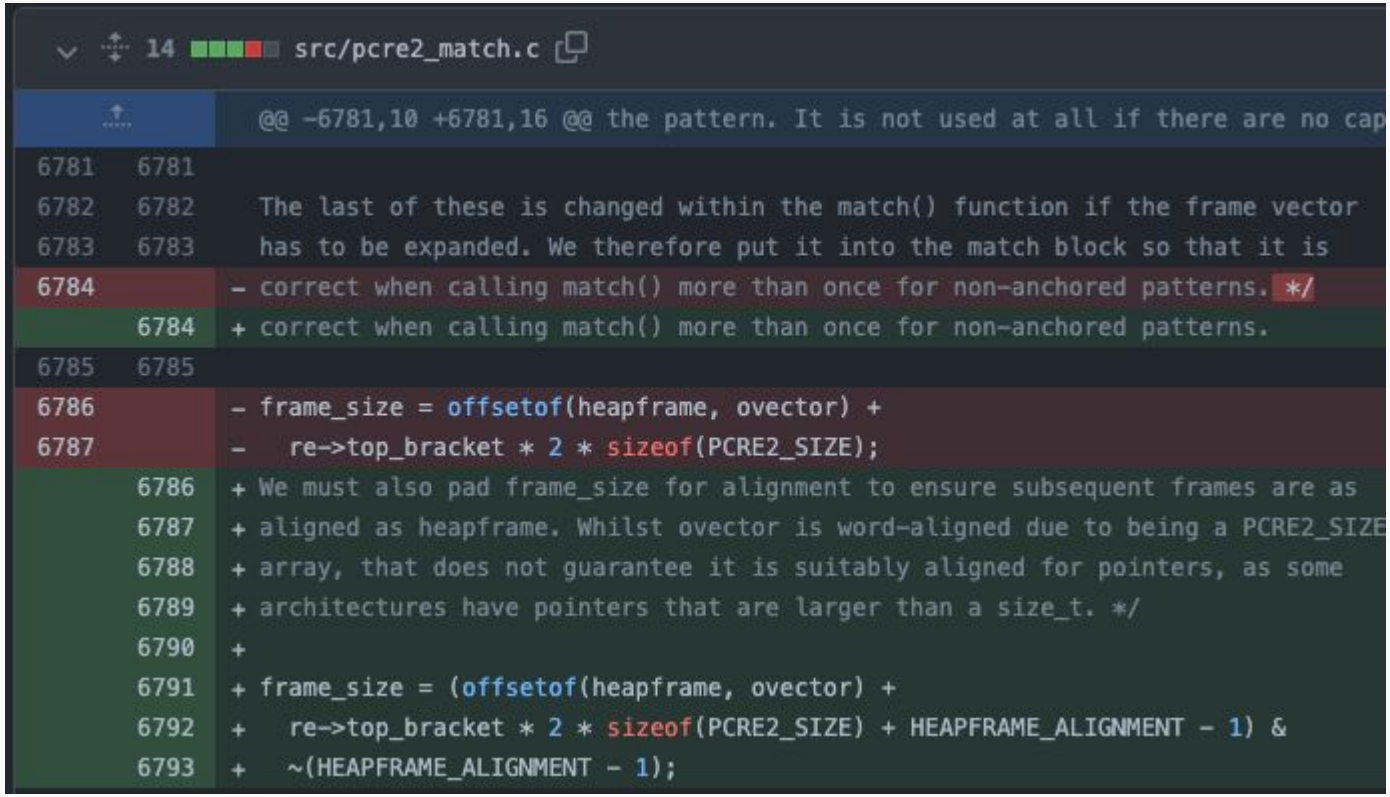


us-mempool.c

```
@@ -149,7 +149,9 @@ _dbus_mem_pool_new (int element_size,
  /* Make the element size at least 8 bytes. */
  if (element_size < 8)
    element_size = 8;
-
+  if (element_size < (int) sizeof (void *))
+    element_size = sizeof (void *);
+
```

[DBus pool allocator]

Incorrectly aligned capabilities



14 src/pcre2_match.c

```
@@ -6781,10 +6781,16 @@ the pattern. It is not used at all if there are no cap
6781 6781
6782 6782   The last of these is changed within the match() function if the frame vector
6783 6783   has to be expanded. We therefore put it into the match block so that it is
6784      - correct when calling match() more than once for non-anchored patterns. */
     6784 + correct when calling match() more than once for non-anchored patterns.
6785 6785
6786      - frame_size = offsetof(heapframe, ovector) +
6787      -   re->top_bracket * 2 * sizeof(PCRE2_SIZE);
     6786 + We must also pad frame_size for alignment to ensure subsequent frames are as
     6787 + aligned as heapframe. Whilst ovector is word-aligned due to being a PCRE2_SIZE
     6788 + array, that does not guarantee it is suitably aligned for pointers, as some
     6789 + architectures have pointers that are larger than a size_t. */
     6790 +
     6791 + frame_size = (offsetof(heapframe, ovector) +
     6792 +   re->top_bracket * 2 * sizeof(PCRE2_SIZE) + HEAPFRAME_ALIGNMENT - 1) &
     6793 +   ~(HEAPFRAME_ALIGNMENT - 1);
```

Insufficient heapframe alignment in PCRE (PCRE_SIZE is only 8 bytes)

Updating pointers after `realloc()`

- The pointer returned from realloc() will have different bounds than the previous allocation (even when growing in-place!)
- Attempting to update any pointers using the previous pointer will give a value that still has the old bounds.
  - For in-place `realloc()`, this will not cover the entire new range (or too much in case of shrinking `realloc()` calls).
  - If the new pointer is not close to the old one, this arithmetic will create a capability that is so far out of bounds that the tag will be cleared.
- Many realloc() calls include this kind of UB, so it's worth auditing calls

Updating pointers after `realloc()`

- Instead of adding a delta, inner pointers must be rederived (example from libX11)

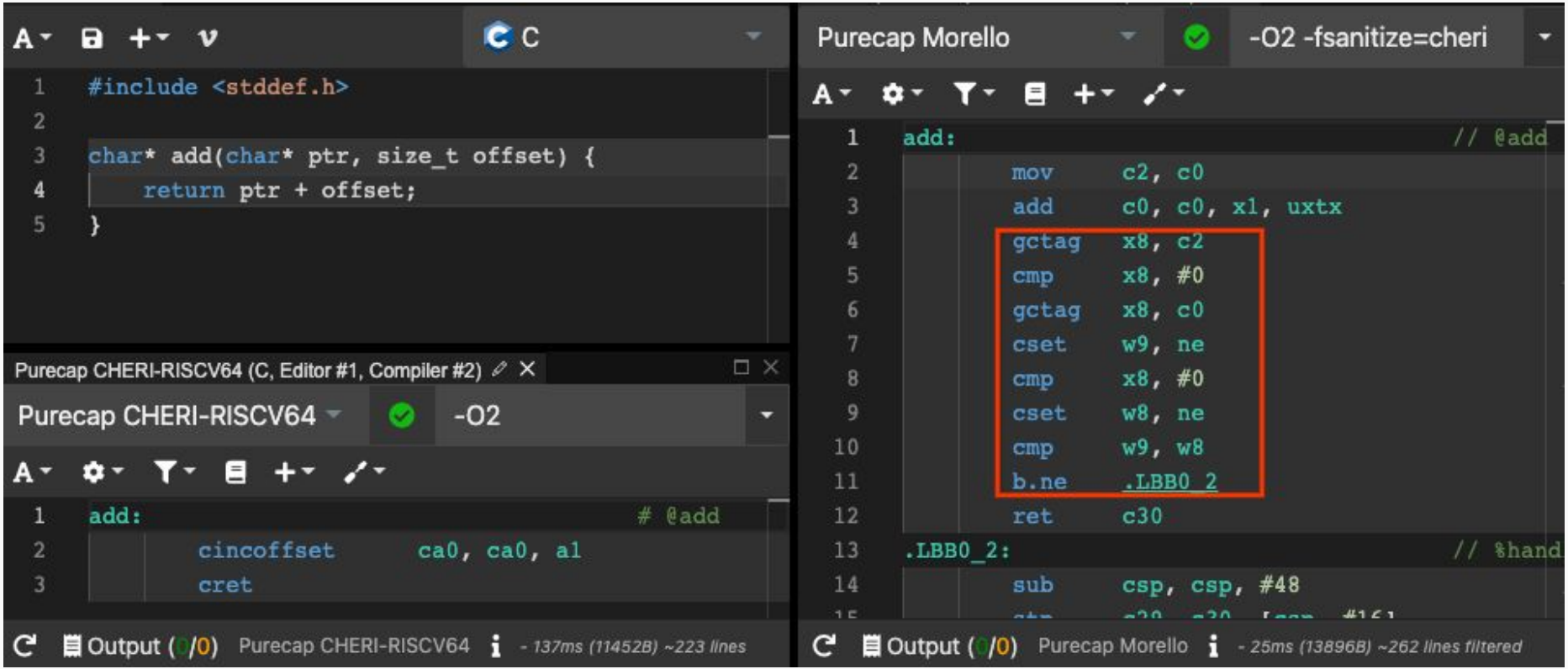
src/xlibi18n/lcDB.c +5 -3 Viewed

```
517 517         }
518 518         if (value != *value_list) {
519 519             int i;
520     -           ssize_t delta;
521     -           delta = value - *value_list;
    520 +           char *old_list;
    521 +           old_list = *value_list;
522 522             *value_list = value;
    523 +           /* Re-derive pointers from the new realloc() result to avoid undefined
    524 +              behaviour (and crashes on architectures with pointer bounds). */
523 525             for (i = 1; i < value_num; ++i) {
524     -               value_list[i] += delta;
    526 +               value_list[i] = value + (value_list[i] - old_list);
525 527             }
526 528         }
527 529
```

CHERI UBSan (`-fsanitize=cheri`)

- To help find places where capability tags are being lost, the CHERI LLVM compiler includes an (experimental) `-fsanitize=cheri` compiler option.
- Instruments all pointer arithmetic to identify where capabilities become unrepresentable.
- Could also be made stricter to identify any non-ISO-C compliant pointer:
  - Only in-bounds and one-past-the-end pointers are legal.
  - Not implemented yet, but is easy to add.
- Often finds updates to pointers after realloc (the difference will often be enough to make capabilities unrepresentable)
- However, there are still false positives:
  - `if (my_uintptr & 1){…}` triggers a false-positive tag loss error.
  - Same when adding a large offset and then casting to a non-capability type.

`-fsanitize=cheri` code generation



Out-of-bounds accesses

Fix out-of-bounds read in FontFileMakeDir()
Alexander Richardson authored 1 year ago

daff8876

```
BuiltinReadDirectory() calls FontFileMakeDir ("", builtin_dir_count); and
this causes the `dirName[dirlen - 1]` access to read before the start of
the string. I found this while porting Xvnc to CHERI-RISC-V (which has
bounds and permissions on all pointers).
```

src/fontfile/fontdir.c +1 -4 Viewed

```
@@ -125,10 +125,7 @@ FontFileMakeDir(const char *dirName, int size)
125 125         dirlen = strlen(dirName);
126 126         attriblen = 0;
127 127     }
128     -   if (dirName[dirlen - 1] != '/')
129     - #ifdef NCD
130     -   if (dirlen)     /* leave out slash for builtins */
131     - #endif
    128 +   if (dirlen && dirName[dirlen - 1] != '/')
132 129         needslash = 1;
133 130     dir = malloc(sizeof *dir + dirlen + needslash + 1 +
134 131                  (attriblen ? attriblen + 1 : 0));
```

- CHERI sometimes detects out-of-bounds accesses that are not noticed otherwise.
- The most common case I have observed is reading beyond bounded buffers derived from string literals [1, 2, 3, 4].
- This happens to work on conventional architectures and with ASan, but fails when buffers are tightly bounded.

(Inline) Assembly Code

- Finally, you may encounter some (inline) assembly.
- Should be very rare – could be manually vectorized code or extremely low-level projects such as kernels or language runtimes.
- Requires the most effort and cannot be diagnosed by the compiler (although it will error on invalid syntax)
- For Morello, in simple cases replacing x-registers with c-registers can be sufficient, but this is highly dependent on the project (partial example from libffi)

```
        /* Partially deconstruct the stack frame.  */
-       ldr     x9, [x29, #32]
-       mov     sp, x9
+       ldr     GP_REG(9), [PTR_REG(29), #4*GP_REG_SIZE]
+       mov     sp, GP_REG(9)
        cfi_def_cfa_register (sp)
-       mov     x2, x29                 /* Preserve for auth */
-       ldp     x29, x30, [x29]
+       mov     GP_REG(2), GP_REG(29)                   /* Preserve for auth */
+       ldp     GP_REG(29), GP_REG(30), [PTR_REG(29)]
```

Porting process overview

1. Pick target project and check if it has already been ported (CTSRD-CHERI GitHub, patches in CheriBSD ports collection, CHERI Slack channels).
2. Run `pkg64 install llvm-base` and try compiling the project.
3. Fix CHERI-specific compiler warnings (unless you are sure they are false-positives).
4. Try running the testsuite and hopefully everything passes (or at least matches the AArch64 baseline, as many projects have test failures on FreeBSD).
5. If not, use GDB to identify where CHERI errors are happening.
6. In case there is a non-obvious missing capability tag, try rebuilding with -fsanitize=cheri and `-Wshorten-cap-to-int`.
7. Failure mode still not obvious? Look for calls to `realloc()` or custom allocators.
8. Still not working? Looks like you may have picked one of the difficult cases… (assembly code, serialization of pointers, misuse of varags, etc.)

Conclusion

- Writing software for CHERI C/C++ **should not require significant changes**.
  - In my experience, updating build systems usually took more time than changing the actual C/C++ code (hopefully not required for CMake/Meson)!
- While there may be some false-positives, fixing CHERI-LLVM compiler warnings is often sufficient to port software.
- If the software fails at run time, GDB will usually locate the cause quickly
  - In case of a missing capability tag, compiling with `-Wshorten-cap-to-int` and `-fsanitize=cheri` will make it easier to identify where the tag is lost.
  - Audit calls to `realloc()` and nested allocators
- If you encounter any further issues that could be diagnosed by the compiler, please file compiler enhancement requests on GitHub :)