Code-centered kernel compartmentalization in CheriBSD

Konrad Witaszczyk, Department of Computer Science and Technology, University of Cambridge, November 2025

The CHERI-extended CheriBSD kernel based on the FreeBSD kernel, as other monolithic kernels, consists of millions of trusted lines of code that are compiled into the kernel binary or separate kernel modules. The kernel binary itself includes code from over 1,700 ELF object files linked together. In this research, I explore compartmentalization of the CheriBSD kernel that focuses on its code (Figure 1) and aims to split the kernel code into compartments that can call functions of other compartments only if a system-defined policy allows such a call.

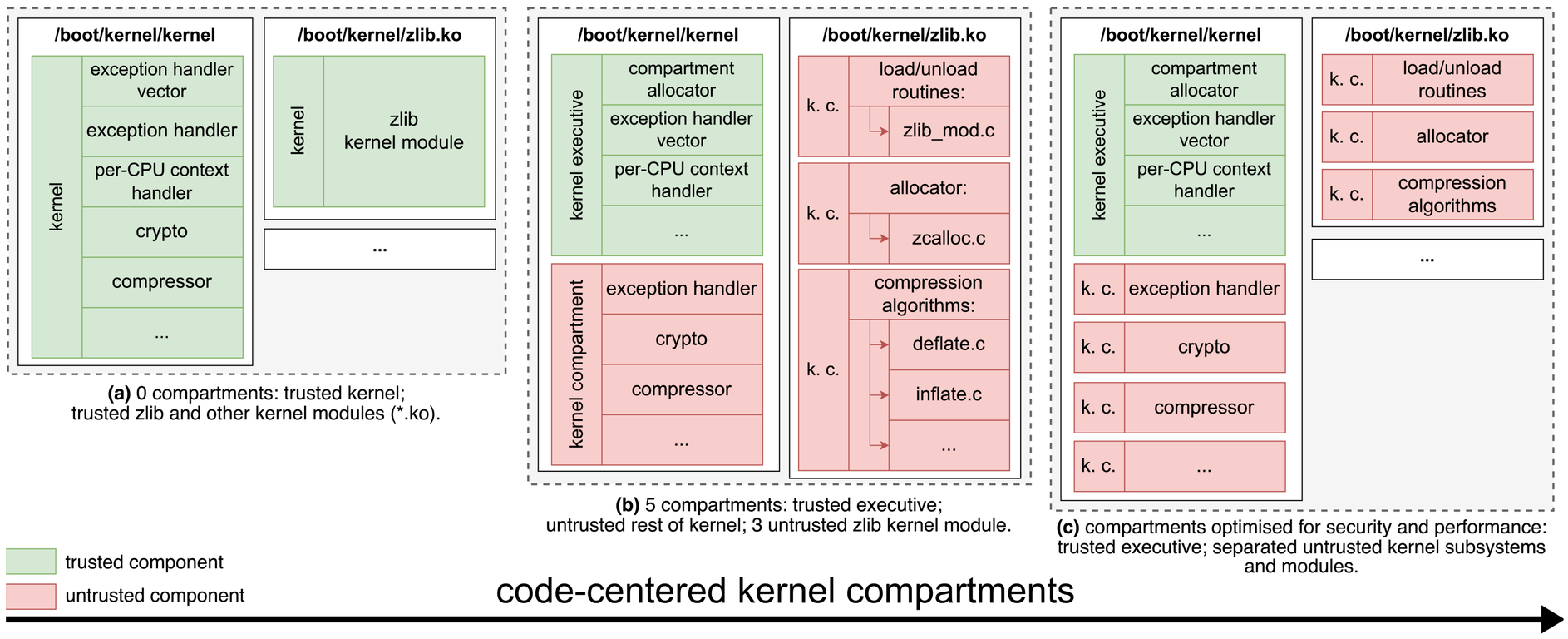

Figure 1. Incremental code-centered compartmentalization approach in a monolithic kernel.

Static and dynamic linking for compartmentalization

The kernel and kernel modules are statically linked into ELF binaries that consist of object files grouped into compartments according to a JSON policy (Figure 2). A module without an explicit policy is linked into a single compartment by default. When relocating symbols at run time, the dynamic kernel linker wraps them with trampolines that implement compartment switching (Figure 3).

```
{
  "compartments": {
        "module": { "files": ["zlib_mod.o"] },
        "allocator": { "files": ["zcalloc.o"] },
        "zlib": { "files": ["deflate.o", ...] }
    }
}
```

Figure 2. An example compartmentalization policy for the zlib kernel module. Case b in Figure 1 makes use of this policy.

Optimal compartment boundaries

The security and performance characteristics of the system can significantly differ depending on the choice of the Trusted Computing Base and untrusted compartment boundaries in a policy. The policy must not separate from the TCB symbols that are required by trampolines, e.g. locking primitives.

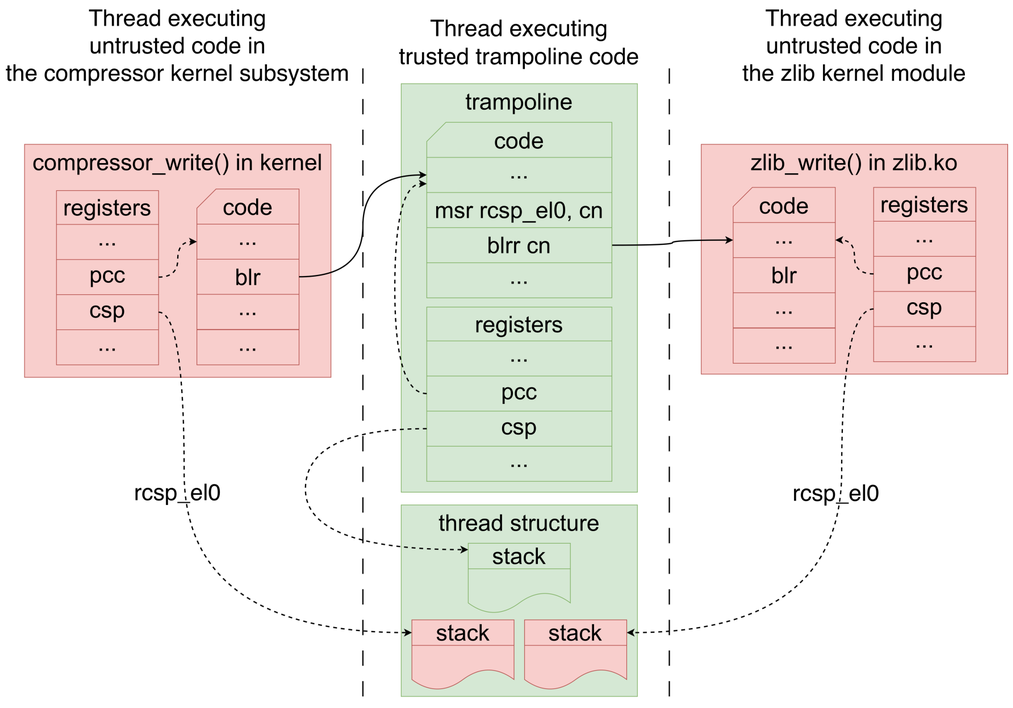

Figure 3. An example function call between compartments using a trampoline.

Supported features

The current prototype (Figure 4) can handle complex workflows on Arm Morello, e.g. the graphics stack (KDE Plasma 5) using DRM and GPU drivers. A compartmentalized kernel can be used together with library-based compartmentalization and temporal safety in the user space. DTrace support is currently being explored.

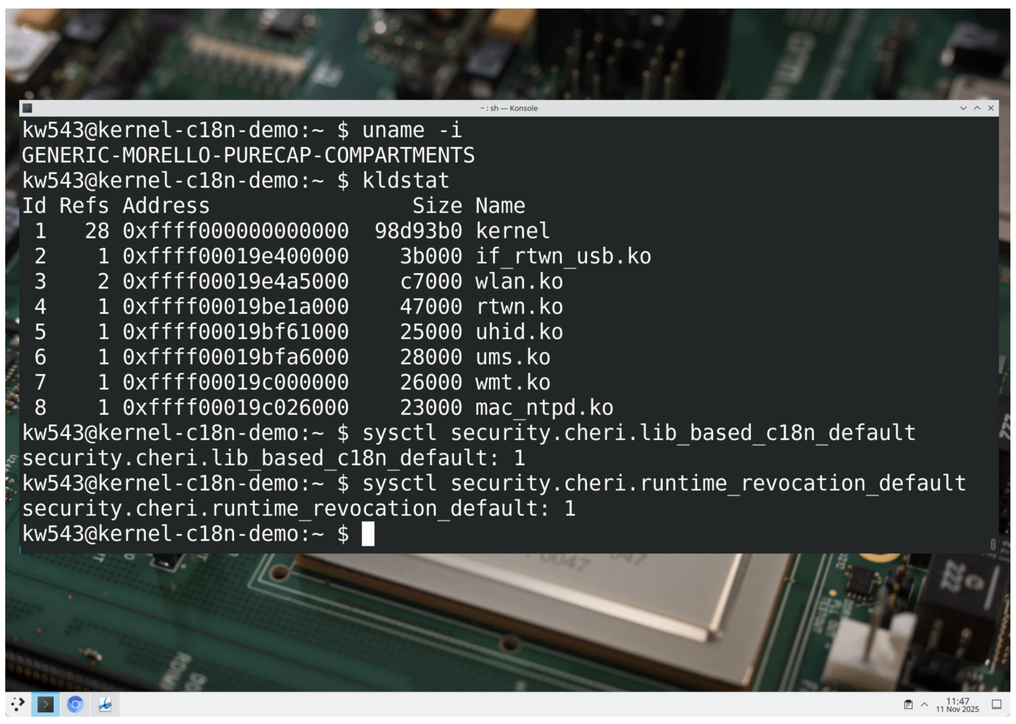

Figure 4. Compartmentalized CheriBSD kernel running on the Arm Morello platform.

Contact Konrad.Witaszczyk@cl.cam.ac.uk

UNIVERSITY OF CAMBRIDGE

Approved for public release; distribution is unlimited. Sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-24-C-B047 (“DEC”) as part of the I2O CPM research program. The views, opinions, and/or findings contained in this report are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.