Lightweight specification and checking of systems code

Ben Simner, Kayvan Memarian
University of Cambridge

March 13, 2026

Ensuring systems code meets its intended security guarantees remains a major challenge. Conventional testing remains standard engineering practice for providing confidence in the correctness of systems. Testing is, however, limited in scope and effectiveness. Full formal verification, while now feasible in some contexts, is often unattainable for the kinds of legacy systems we see in production. We revisit a middle ground, so-called lightweight verification, with the expressiveness of full formal verification for much less effort, but with only partial guarantees. We investigate this along two axes: a top-down approach writing separation-logic-style specifications for a large fragment of a production system, with lightweight runtime checks; and a bottom-up approach, dynamically checking page table manipulation code for low-level errors. We exercise these respectively on two production hypervisors for Arm, on Android (pKVM) and on FreeBSD (bhyve).

For the former, we write specifications – in the source language – for much of the top-level hypercall API of pKVM covering the functional correctness, ownership, and 2 phase-locked concurrency, and check them in a lightweight way at runtime. We do this by writing computable abstraction functions which return an abstract *ghost state* from the concrete machine one. This allows one to write specifications as morally pure C functions over the ghost data structures, without requiring specialist tools or languages This redundancy, from an abstract re-implementation of the system, helps not only in the production of the spec (i.e. working out *what it should be*), but also provides a naturally computable check that the system conforms (by comparing the abstract and concrete states). We then write a custom test suite and rudimentary fuzzer to exercise pKVM, and discover a small collection of bugs which were reported. This was published at SOSP'2025, but we are exploring using existing proof-based tools in a more lightweight way, and balancing that with the increased required specialist skills.

For the latter, we produce a provably-sound abstraction of the Arm architecture, encoding a pagetable-update protocol software must follow; we reify as a state machine baked into a runtime monitor (called *Casemate*), which we compile into pKVM and bhyve. Casemate then automatically detects unsynchronised races to the pagetables, insufficient TLB management, e.g. from violating the Arm break-before-make requirements, and a variety of other low-level errors, at runtime. We then run various test loads with the monitor enabled, uncovering a number of corner cases, including at least one security-critical bug in pKVM. This work is currently in-progress, and we are investigating how it can be applied more widely or extended in other directions.

Both of these integrate well with existing testing-oriented workflows used by developers today, and enhance those workflows with richer properties. Our focus on existing legacy systems not only benefits the systems we have today, but proves developers can and should employ lightweight verification methods for their own codebases – whether or not written by them, or even not by a human.

We report on our experiences, the current progress on each task, and our future plans.