Hints for High-Assurance Cyber-Physical System Design

Lee Pike
Galois, Inc.
leepike@galois.com

May 18, 2016

Butler Lampson published "Hints for Computer System Design" in 1983 that describes sound engineering practices for building large and complex computing systems. More than 30 years later, the advice is still relevant yet sadly ignored. With deference to Lampson, I provide hints for building high-assurance cyber-physical systems (CPS). While Lampson broadly considers computer systems, ranging from programming languages to the (beginnings of) the internet, I focus on security- and safety-critical systems; examples include aircraft, automobiles, and embedded medical devices. The domain obviates some of Lampson's advice; for example, Lampson notes that "one crash a week is usually a cheap price to pay for 20% better performance", but I cannot endorse such advice for software that puts peoples' lives at stake. Moreover, high-assurance systems are often embedded systems, and the advice focuses on that domain.

The hints are drawn from high-assurance systems I have been involved in designing and building, both in government and industry. For the sake of continuity, most of the examples I draw on come from DARPA's *High-Assurance Cyber Military Systems* (HACMS) program. The goal of the program is to demonstrate the feasibility of building "hack-proof" software for complex cyber-physical systems. The problem space is motivated by the insecurity of modern CPS software, such as automotive software. The approach of HACMS is to leverage advances in formal verification to *guarantee* a system's correctness.

My team, in collaboration with others on the program, focused on building secure autopilot software for an unpiloted air vehicle (UAV), called *SMACCMPilot*. Our deadline was tight: with a team of three engineers and an eighteen month deadline for the first operational release, we built new programming languages from scratch, and then used them to build an autopilot. An autopilot is a bit of an understatement since we built a full system of which the core autopilot was one part: we built a board support package for new hardware, device drivers, secure wireless communications with a base station, arming logic, motor controller, control loops, etc.

To assess our progress, not only did we provide live flight demonstrations, but an experienced "red team" was given full access to the systems, including source code, and they attempted to discover vulnerabilities. Mostly, they did not. As one government official said, our team "had likely built the most secure UAV in the world."

While I will give concrete examples from HACMS to illustrate my hints, this paper does not describe how to build an autopilot or the research we undertook. This paper is about how the general lessons we learned can be applied to future systems. Furthermore, this paper does not focus on formal verification despite HACMS being primarily a formal verification project. None of the hints require esoteric verification expertise.

I give five hints: (1) *Constrain the programming languages*, (2) *Secure the interfaces*, (3) *Automate the tedium*, (4) *The Verifier's Dilemma*, and (5) *The mythical verification month*.
In addition, I talk about our failures, for which others can hopefully provide hints in the future.

As Lampson said, these are just hints, not laws, rules, principles, etc. There are exceptions and counterexamples.