White-box Software Isolation with Fully Automated Black-Box Proofs

Motivation

- Software Isolation: No new or unintended behaviors can be introduced via external inputs
- Software Isolation important for:
  › Safely running untrusted code in trusted host, e.g., running ad hoc crowd-sourced code
  › Preventing subversion of critical systems, e.g., medical devices, avionics systems
- Since Software Isolation is important, we want:
  › Strong evidence that critical software is isolated
  › Programmer-evident isolation mechanisms
- Our Approach:
  › Formal verification: Provides strong evidence
  › Isolation mechanisms: Must be in source-code

Goals / Scope

- Goals:
  › Enable white-box software isolation: No post-compilation modifications required
  › Enable black-box software isolation proofs:
    • Fully automated proofs
    • No specialized inputs, e.g., loop invariants, function pre-/post-conditions
- Scope: ARM machine-code programs
  › ARM: Dominant mobile/embedded platform: Many critical applications
  › Machine-code: Minimizes Trusted Computing Base (TCB): Excludes compiler from TCB

Architecture

Design and Implementation

- Software Isolation: Memory & Control-flow Safety
  › Programmer-evident software isolation: Isolation mechanisms are programmer-visible
- Automated Safety Property Verification
  › Input: Machine-code of program
  › Output: Safety proof, or proof failure addresses
- Approach:
  › Extends Hoare Logic for ARM machine-code [Myreen'07] to reason about safety properties
  › Abstract Interpretation: Automated proof obligation discharge with failure termination
  › Developed logic framework: AUSPICE [Tan'15]
  › Implemented in HOL4 theorem prover
