Toward a Verified, Secure, General-Purpose Microkernel

Jonathan S. Shapiro, Eric Northup, M. Scott Doerrie, Swaroop Sridhar
Johns Hopkins University

With particular thanks to:
Hao Chen, David Wagner (UCB)
Matt Kauffman, J. Strother Moore (U.T. Austin)
Bill Bevier (AMD, Inc.)
Steve Crocker (Shinkuro, Inc.)
Quick Review

You have:
- A set of security, isolation requirements
- A model of a system

You want to know:
- Does the system you built meet the requirements?

Approach:
- Verify that the operational semantics of the model satisfies the requirements (Shapiro & Weber, 2000)
  - Must formalize requirements (goals)
  - Must formalize model
- Verify correspondence: does implementation match the model.

Sufficient rigor is moderately hard, but tractable.
Complications

- Sufficient rigor is hard.
- Need an implementation language that you can reason about formally.
  - Usually assumed that aliasing needs to be restricted
    - no general pointers!
  - We found an alternative
- From a practical standpoint, need to use a standardized language
  - That leaves Ada
- But after you hire all of the surviving ADA programmers...
Traditional Approach

Source Program (Ada or C) → Compiler → Binary
Traditional Approach

Source Program (Ada or C) → Importer → Program Model

Compiler

Binary
Traditionally, the approach involves:

1. **Source Program (Ada or C)**
2. **Compiler**
3. **Binary**
4. **Importer**
5. **Program Model**
6. **Goal Theorems (Requirements)**
7. **Prover**
8. **Language Semantics Model**
9. **Satisfaction?**
Traditional Approach

Source Program (Ada or C) → Compiler → What You Run

Importer → Goal Theorems (Requirements) → Prover → Language Semantics Model

What You Proved → Program Model

Satisfaction ? → What You Assumed
Traditional Approach

Source Program (Ada or C)

What You Run

Compiler 485 Klocs

What You Assumed

Binary

What You Proved

Importer O(60 Klocs)

Program Model

Satisfaction?

Goal Model and Theorems (Requirements)

Prover

Language Semantics Model

O(400 ISPages)

ISPPage: a page of international standardese
BitC Approach (Interim)

Target Program (C) -> Compiler 485 Klocs -> Source Program (BitC) -> What You Proved

Exporter O(100 lines) -> Inspected

What You Run -> Binary
BitC Approach (Interim)

**Target Program (C)**

**Exporter O(100 lines)**

**Source Program (BitC)**

**Compiler 485 Klocs**

**What You Run Binary**

**What You Proved**

**Goal Model and Theorems (Requirements)**

**ACL2 Compiler**

**Inspected**

**Prover**

**Verified!**

**Satisfaction?**
BitC Approach (Eventual)

- What You Assumed
- Source Program (BitC)
- Machine Model
- Goal Model and Theorems (Requirements)
- Binary
- Prover
- What You Run
- Native Compiler O(???)
- What You Proved
- ACL2 Compiler
- Verifiable
- What You Assumed
- O(10 Klocs)
- Prover
- Satisfaction ?

Verifiable

Verified!
The Good News

- EROS is pretty easy to specify.
  - Atomic units of operation: it's really just a big state machine
  - The externally visible abstractions are relatively easy to formalize (address spaces, processes)
- We can duck the aliasing issue because the implementation can (and does) restart system calls when it gets into a corner.
- From prior work, we think we know what properties we are trying to prove.
- EROS-NG is much simpler and faster than EROS
The Good News

- EROS is pretty easy to specify.
  - Atomic units of operation: it's really just a big state machine
  - The externally visible abstractions are relatively easy to formalize (address spaces, processes)
- We can duck the aliasing issue because the implementation can (and does) restart system calls when it gets into a corner.
- From prior work, we think we know what properties we are trying to prove.
- EROS-NG is much simpler and faster than EROS Secret Sauce!
Things We Know How to Verify (We Think)

- All required access checks actually happen.
- No TOCTOU errors
- Every kernel path terminates in bounded time.
- Correctness of address translation and page table invalidation.
- Correctness of states (e.g. stopped process cannot receive)
- Correctness of dependency invariants
- Enforcement of confinement preconditions
- Correspondence to the abstract operational semantics (as revised).
- *(BitC is inherently memory safe)*
**End Result**

- First general-purpose, fully verified security kernel
- And oh yes:
  - Still fast
  - Still real-time
  - Still embeddable
  - Still runs on commodity hardware
  - Subject to secure boot assumptions
- But also:
  - First generally available verification infrastructure for systems programmers
  - Identification of a class of important programs that we can actually verify things about (atomic transactional).