Safe Kernel Extensions
Without Run-Time Checking

George C. Necula
Peter Lee
The Problem

Code from **Untrusted Sources**
Your Options

- Hardware-Assisted Protection
- Software Fault Isolation
- Safe Languages and Runtime Environments
Wishlist

• easy validation of untrusted code
• execution with no runtime overhead
• no cryptography
• no trusted third party
• no program analysis
• no code editing, compilation, interpretation
• code consumer publishes safety policy

• code producer compiles program and certifies its adherence of the policy

• PCC-binary contains native code and safety proof

• code consumer validates the proof

• native code can then run at full speed
Safety Policy

What behavior is considered safe?

1. verification-condition generator
   • procedure computing a predicate from code
   • design can be simplified with an abstract machine

2. precondition
   • calling convention

3. axioms for validating the predicate
   • inference rules
Proving the Predicate

• inference rules of first-order predicate calculus plus some register arithmetics

• given the precondition we must infer the verification condition (VC)

• calculus gives that VC holds

• safety theorem gives that the code is safe according to the policy (proof available)
PCC binary

- contains native code and binary representation of the proof
- can be tampered with
  - if you change the code and it is unsafe, the validator will notice
  - if the validator does not notice, your code is safe (by safety theorem)
Validating the Proof

- calculate the safety predicate from the given code with generator from the policy
- check if the proof’s result is the safety predicate
- check if the proof’s assumed preconditions match the ones in your policy
- check, if all steps in the enclosed proof are valid instances of inference rules
Evaluation

- safe code execution
- as fast as native code
- versatile policies possible
- extra runtime cost for validation
- binary size increases
- proving step is hard
Imagine ...

- protection with object granularity
- no application speed degradation
- we could check for locks, secrets, time
- could be mitigated by code signing?
- no problem?
- certifying compilers?