CIS 890: Language Based Security

Torben Amtoft and Anindya Banerjee

Kansas State University

Fred B. Schneider and Greg Morrisett and Robert Harper:
A Language-Based Approach to Security
Informatics—10 Years Back, 10 Years Ahead, LNCS 2000

Goal: Leverage program rewriting and program analysis to enforce security policies

Enduring Design Principles

Minimal Trusted Computing Base

Enduring Design Principles

Minimal Trusted Computing Base but now kernels are huge (for reasons of efficiency)

Enduring Design Principles

Minimal Trusted Computing Base but now kernels are huge (for reasons of efficiency)

Principle of Least Privilege

Enduring Design Principles

Minimal Trusted Computing Base but now kernels are huge (for reasons of efficiency)

Principle of Least Privilege but now almost all code runs on behalf of single user

Enduring Design Principles

Minimal Trusted Computing Base but now kernels are huge (for reasons of efficiency)

Principle of Least Privilege but now almost all code runs on behalf of single user

Violation of principles provides excellent conditions for vira: scripts hidden in emails run with all the privileges of the user that receives the message.

Enduring Design Principles

Minimal Trusted Computing Base but now kernels are huge (for reasons of efficiency)

Principle of Least Privilege but now almost all code runs on behalf of single user

Violation of principles provides excellent conditions for vira: scripts hidden in emails run with all the privileges of the user that receives the message.

Principles can no longer be ensured by operating systems, but can be supported by Language-based security

Reference Monitors

Examples

- access control
- traps for system calls

Trade-off between effectiveness and protection from subversion

- Reference monitors can always be simulated by software
- But an analyzer operating on program text more powerful than operating on single execution only.

Security Policies

Implementable by reference monitor?

- Yes: Access control

Security Policies

Implementable by reference monitor?

- Yes: Access control
- No: information flow, availability

Security Policies

Implementable by reference monitor?

- Yes: Access control
- No: information flow, availability

Requirements:

- acceptable sequences prefix closed
- non-acceptable sequences rejected after finite number of steps
- sufficient to examine one sequence (i.e., a property)

Whether information flows from x to y: is not a property:

`public` := 1; **if** `secret` = 7 **then** `public` := 0

Sequence (`public` := 1) seems benign, but is not!

Language based approaches

- Program Rewriting (inlined execution monitors)
- Program Analysis
- Certifying Compilation

Inlined Execution Monitors

Rewrite incoming code by inserting checks (must ensure that target program cannot bypass these)

One most define

- security events
- security state
- security updates

Method is compositional.

For sensible inlining, some structure and/or annotation must be present

Type Systems

Shifts burden of proof from recipient to consumer

Well-typed programs do not go wrong
for instance, cannot apply wrong operations to wrong values

Run-time checks still needed for array bounds etc., in the absence of stronger systems (like dependent types)

Certifying Compilers

Compiler produces object code equipped with proof

Certificate checker part of trusted base, compiler not!

Types as proofs (until recently, type systems for high-level languages only):

- JVM code (but policy supported very weak)
- TAL (Typed Assembly Language)

Most aggressive and general: PCC (Proof Carrying Code)

Synergy among approaches

Adding IRM, checking for say array bounds, to

- type systems

Synergy among approaches

Adding IRM, checking for say array bounds, to

- type systems These can check complex properties like information flow, but it is cumbersome to check for array bounds

Synergy among approaches

Adding IRM, checking for say array bounds, to

- type systems These can check complex properties like information flow, but it is cumbersome to check for array bounds
- certifying compilers These can also check complex properties, but it may be burdensome to prove at compile time that an index gets never out of bound.

More synergy

IRM + certifying compilers

- rewriting should be followed by optimizations.

More synergy

IRM + certifying compilers

- rewriting should be followed by optimizations.
- would not like optimization routines to be part of trusted base.

More synergy

IRM + certifying compilers

- rewriting should be followed by optimizations.
- would not like optimization routines to be part of trusted base.
- Insert proofs that removed checks are not needed!