THE CASE FOR LANGUAGE-BASED SECURITY

F B Schneider

Rapporteur: I S Welch

The Case for Language-Based Security

Fred B. Schneider
Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.

Overview of the Case

- Security needs are not being met.
  - Extensible systems. (WWW a special case)
- "Programming Languages" offers solutions.
  - Automatic program analysis (e.g. type checking)
  - Automatic program transformation (e.g. optimization)
  - Automated deduction (e.g. proof checking)

Extensible Systems

Increasingly, systems have programmable or extensible interfaces:

- printers and editors (postscript printers, emacs, Word)
- browsers and servers (applets, plugins, CGI-scripts)
- application automation (macros, VB scripts, activeX)
- operating systems (plug-and-play, virus scanners)
- networks (firewalls, active networks, JINI)

2

Why Extensible Systems?

In favor of extensible systems:

- Resource optimization (client-side checking)
- Preservation of market-share (easy to add features)
- System maintenance and evolution (software subscriptions)
- Programming the networked world (dynamic discovery/access)

Building an extensible systems makes good economic and engineering sense.

Clear trend in COTS and elsewhere.

3

Why Not? Security!

The obvious solution:

- Rule out misbehavior (="enforcement")
- Opportunities for deploying enforcement mechanisms:
  - **Before** execution (program analysis)
  - **At** execution (program transformation)
  - **During** execution (program monitoring)

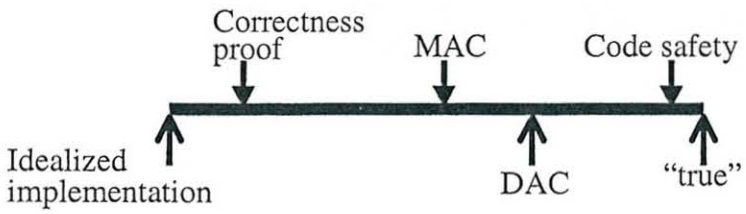
4

Ain't Misbehaving?

Whether it's misbehavior depends on:

- The task -- what is its purpose?
- The platform executing it.
- Some given security policy.

Tighter constraints on behavior are better!



5

Principle of Least Privilege

> Each principal is given minimum access needed to accomplish its task. [Saltzer & Schroeder 1975]

Power from richer notions of "principal" and of "minimum access":

- granularity of objects/operations/accesses
- application or system state
- application or system history
- platform environment...
  E.g., time, location, available power, bandwidth

6

Central Research Challenge:

"Do" Principle of Least Privilege

- How to do "enforcement" in accordance with Principle of Least Privilege?
  - Efficiently
  - With TCB that is small and is independent of language and operating system.
- How to synthesize "minimum access"?
  - Despite refinement and interface hiding.
  - Without much (any?) help from users.
  - Without much (any?) help from programmers.

7

The Case for Language-Based Security

- Extensible Systems
- Security needs of Extensible Systems
- Language-based enforcement of security policies.

8

"Doing" Principle of Least Privilege:

Language-based Enforcement

- Inlined Reference Monitors (IRMs) [Schneider]
  - Generalizes software fault isolation (SFI).
  - In principle, can enforce any EM security policy.

  ... monitors execution and blocks "bad things".
- Certifying Compilation [Necula & Lee, Morrisett]
  - Generalizes type checking.
  - In principle, can enforce any security policy.

  ... moves TCB from a code producer to a proof checker.

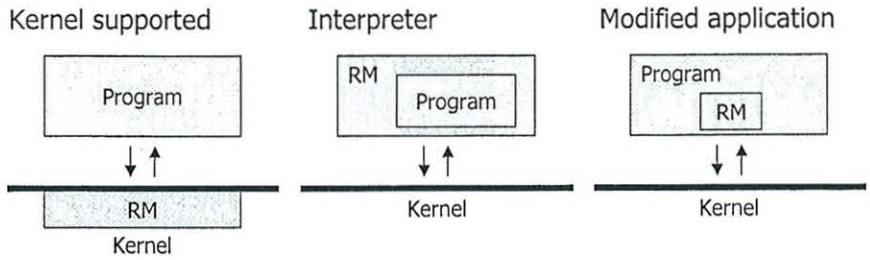
9

Reference Monitor

Monitor execution to prevent bad behavior.

Implementation requires ...

- Capturing policy-relevant events
- Protecting reference monitor from subversion



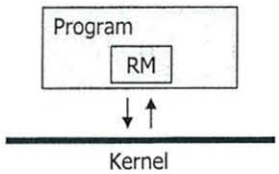
10

Language-based Enforcement:

In-lined Reference Monitors

When mechanism inserted into the application ...

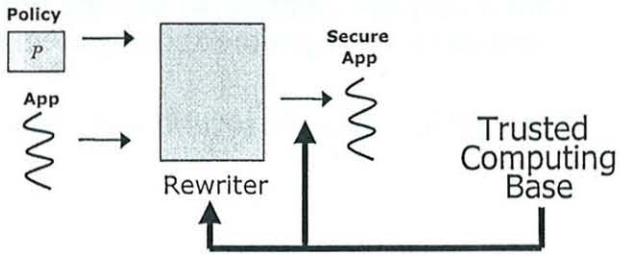
- Allows policies in terms of application abstractions.
- Pay only for what you need.
- Enforcement without context switches into kernel.
- Isolates state of enforcement mechanism.



11

Building an IRM

- Fundamental issues:
  - Does the application behave the same?
  - Can the application subvert the reference monitor?
- Pragmatic issues:
  - What policies can be enforced?
  - What is the overhead of enforcement?



12

What Policies can IRM Enforce?

Class EM enforcement mechanisms:
Monitor a target system and terminate any execution that is about to violate the security policy.

EM includes reference monitors and other operating systems and hardware-supported mechanisms.

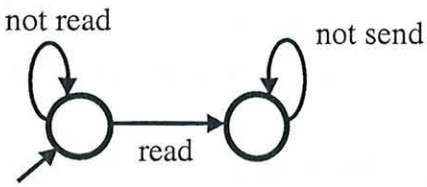
EM allows ...

Principle of Least Privilege: Each principal is given minimum access needed to accomplish its task.

13

What polices can IRM enforce?

Security automata



Thm: Every EM enforceable security policy can be characterized by some security automaton.

Thm: EM enforceable security policies ⊆ safety properties.

14

Definitions

Security policy P: predicate on sets of executions

Target system S: set Σ of executions

S satisfies P: $P(\Sigma)$ = true

Warning: Don't expect

($\Pi \subseteq \Sigma$ and $P(\Sigma)$) implies $P(\Pi)$

to hold.

15

What polices can IRM enforce?

Characteristics EM enforceable

Any EM enforcement mechanism ...

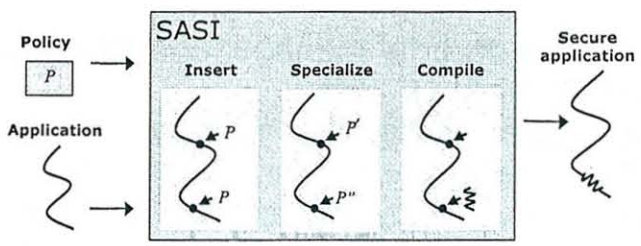
- Analyzes the single (current) execution.
  $$P(\Pi): \quad (\forall \sigma \in \Pi: \ p(\sigma))$$
- Must truncate execution as soon as prefix violates policy: $\neg p(\tau) \Rightarrow (\forall \sigma : \neg p(\tau\,\sigma))$
- Must detect violations after a finite time:
  $$\neg p(\sigma) \Rightarrow (\exists i : \neg p(\sigma[..i]))$$

EM policy implies safety property

16

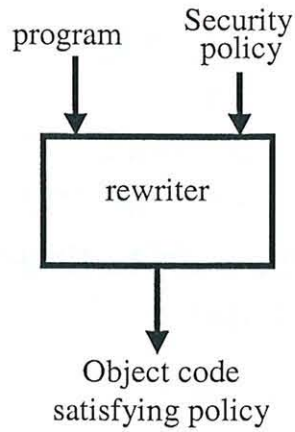
Enforcement Overhead?

- Insert check before each machine instruction.
- Eliminate unnecessary checks by partial evaluation (using local knowledge about insertion point).



17

Initial Prototype: SASI



Handles:
- gcc output for x86
- Java VM

18

Evaluation of SASI prototypes

- Reproduce MiSFIT [Small '96] functionality for x86 with SASI.
  - SASI clearer: 2 page automaton specification.
  - SASI sometimes more expensive: SASI doesn't change code... just inserts instructions.
- Delete Java security manager and run only SASI'd applets and SASI'd library.
  - SASI clearer: 5 page automaton specification.
  - SASI can be slightly cheaper: deletes calls and checks.
  - SASI more expressive: checks can be anywhere.
  - Java SM: No added code or c-time overhead.

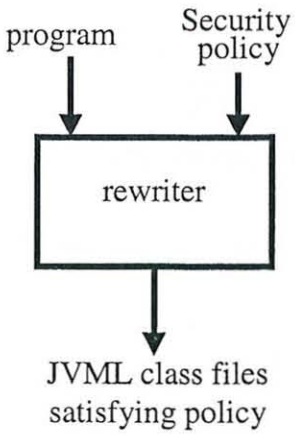
19

Lessons Learned from SASI

- Checking machine instructions problematic:
  - Applications use abstractions (e.g., methods)
  - Must add code for some events (e.g., interrupts)
  - Must synthesize policy-level events
  - ➔ Specifying policies requires general computation and structured state
- Can build on language-based guarantees
  - Also works for x86: Typed Assembly Language, ECC, ...

20

Second Prototype: PoET/PSLang



Policy Enforcement Toolkit
Policy Specification Language

... handles JVML class files

21

PSLang:

Policy Specification Language

- Subset of Java. (TCB is thus small)
- Event-driven programming model binds program actions (events) to security state updates.
- Policy expressible using application abstractions.

```
ADD STATE { boolean did_read = false; }

ON EVENT methodCall FileInputStream.read { did_read = true; }

ON EVENT methodCall Network.send CONDITION did_read { FAIL; }
```

22

PoET:

Policy Enforcement Toolkit

- Routines for accessing state and events.
- Rewrites JVM class files and eliminates redundant checks. (17.5K loc = TCB)
- Program must obey certain constraints so code inserted works properly:
  - JVM verifier already provides necessary guarantees!
- Allows policy composition (conjunction):
  - Used in Prism Digital Library project for "loaning" digital objects. Manages security and preservation policies.

23

Classic Secuirty Principles

Principle of Least Privilege.

Each principal is given minimum access needed to accomplish its task.

Minimal Trusted Computing Base.

Assurance increases when a mechanism is small and simple.

24

Language-based Enforcement:

Certifying Compilation

Writing proofs is **hard** but checking proofs is **easy**.

Questions needing answers:

- Did IRM rewriter add correct checks?
- Did optimization of IRM introduce vulnerabilities?
- Are type checker claims correct?
- Does program satisfy specified policy?

25

Certifying Compilation:

Reducing TCB size

- Remove analyzer from TCB by requiring it to produce a "proof" substantiating its conclusions.
- Add "proof checker" to TCB.

TCB := **TCB** – **analyzer** + checker

26

Certifying Compilation:

Example: Proof carrying code (PCC)

General Approach: Execute only "certified code" which can be checked automatically.

How PCC works:

1. "Code + Proof" sent to host.
2. Host checks proof against code and policy.
3. If proof checks then host runs program.

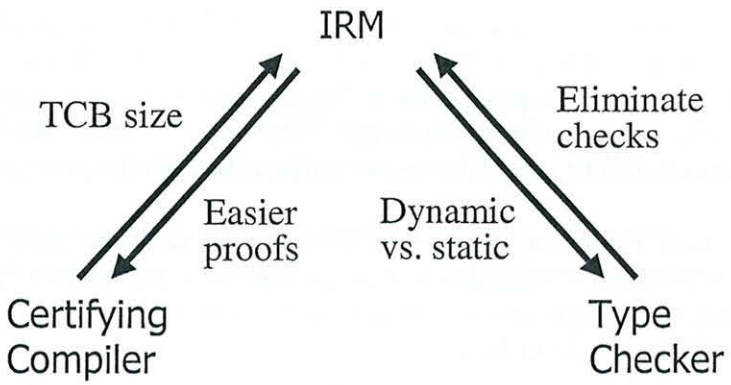
Proof produced automatically for some policies.

- Adding code (explicit checks) facilitates producing proof.

27

Summary

Elements of Language-Based Security:



28

References:

- Fred B. Schneider. Enforceable Security Policies. *ACM Transactions on Information and System Security 3*, 1 (Feb. 2000), 30-50.
- Ulfar Erlingsson and Fred B. Schneider. SASI enforcement of security policies: A retrospective. *Proceedings of the New Security Paradigm Workshop* (Caledon Hills, Ontario, Canada, September 22-24, 1999), Association for Computing Machinery, 1515 Broadway, NY, NY, 87-95.
- Ulfar Erlingsson and Fred B. Schneider. IRM enforcement of Java stack inspection. *Proceedings 2000 IEEE Symposium on Security and Privacy* (Oakland, California, May 2000), IEEE Computer Society, Los Alamitos, California, 246-255.

29

DISCUSSION

Rapporteur: I S Welch

Lecture Two

Professor Randell asked if Professor Schneider had considered describing the cost of the constraints on behaviour described by a security policy.

Professor Schneider replied that it was important to distinguish between the policy and enforcement. At the moment he was just talking about policy and argued that it is more difficult to perpetrate an attack if the policy rules out more behaviours. At the present the discussion ignores issues of performance but this an engineering problem of improving performance of the enforcement mechanisms. When talking about performance and policies the real question is usability, which is a matter of how you design your policy.

Dr Horning asked if Professor Schneider could give an intuition about the description used sets. Professor Schneider replied that it was in order to stay completely general. If EM was described in terms of single operations then policies where you need a correlation between sets of operations could not be described.

Professor Randell asked if you can have a security violation from combined effort of a set of principals where each is doing a legal thing. Professor Schneider replied that EM could cope with this if the notion of state included as much of the universe as was necessary to include all the activities of the principles.

Dr Horning asked if Professor Schneider had anything to say about the possibility of tampering with code after it was loaded. Professor Schneider replied that you need to preserve integrity of the reference monitor and with the IRM approach we assume that the operating system somehow prevents the rewriting after loading to take place.

Mr Warne stated that he had got lost about the role of the principal. Here Professor Schneider is talking about programs but where is the role of the principal. Professor Schneider replied that unfortunately many security enforcement mechanisms usually associate principals with the user but this does not prevent certain types of security problems. A virus is a good example of the problem of relying upon principals.

Professor Burns stated that he was concerned that by merely checking the next step Professor Schneider does not have expressive enough a model, for example, with deadlocks, checking the next step is too late. Professor Schneider replied that this illustrated the kinks in the model. But Professor Schneider stated that by analogy to deadlock detection it may be possible to determine the first point at which it goes sour.

Professor Burns asked about a case where you want to ensure that a message is only sent an even number of times. It seems that here you need to look ahead, and you can't know halfway through whether later you are going to be okay. Professor Schneider replied that he wasn't sure

if this was expressible as a safety property and if it isn't a safety property then it cannot be enforced. He went on to discuss what if we imagine a deadline for sending a message, here we don't want to have a policy enforced that results in the program being stopped if the message deadline is violated. The problem here is we haven't got the right concept of system as we imagined it including an absolute clock, but since don't know anything about that so have to talk about the local clock. This problem is related to a term in concurrent programming literature called unrealizability.

Another participant asked what about corrupt input messages? Professor Schneider replied that this question sounded like "what would be a good security policy to put into the input message handler?" So these questions become what are the correct policies to add to each component of the system. This is an important question that should not to be solved in an ad-hoc way, maybe something like Professor Dijkstra's weakest precondition calculus would allow us to infer least privilege. However this is not necessarily practical for a non-trivial system and so system security programmers will develop knowledge of what to protect.

Mr Anderson suggested that the enforcement mechanism is not effective against collaboration unless you can look at the state and then you come up against problems of observability. Professor Schneider asked if Mr Anderson was thinking of a distributed system. Mr Anderson replied that he could imagine problems even when considering a shared memory model because of multiple threading. Professor Schneider replied that yes this is a visibility problem although the question really is can you always decompose a policy into a set of automata whose conjunction defines the policy. The answer is no but one can define each individually and IRM in another policy into another policy thereby getting the conjunction.