Intrusion Detection via Static analysis

David Wagner
U.C. Berkeley
daw@cs.berkeley.edu

Drew Dean
Xerox PARC
ddean@parc.xerox.com

Background

Anomaly detection:

- Develop a *model* of normal program behavior.
- Flag any deviations from the model as potential intrusions.

$\Rightarrow$ Can detect *unforeseen attacks*.

The Problem

> Often, [intrusion detection] systems deliver such a high number of ‘false positives’—which classify an action as an intrusion when it may be legitimate—that computer operators ignore intrusion alarms altogether. (InternetWeek, 1999)

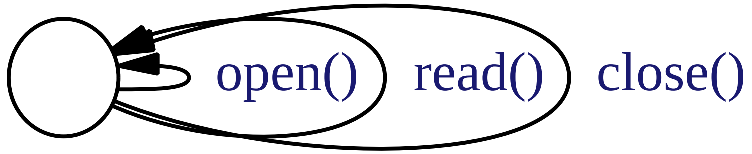
Challenge: Find models that reduce the false alarm rate.

The New Idea

Deduce a model from program source code, using static analysis.

(compare to prior *machine learning* work)

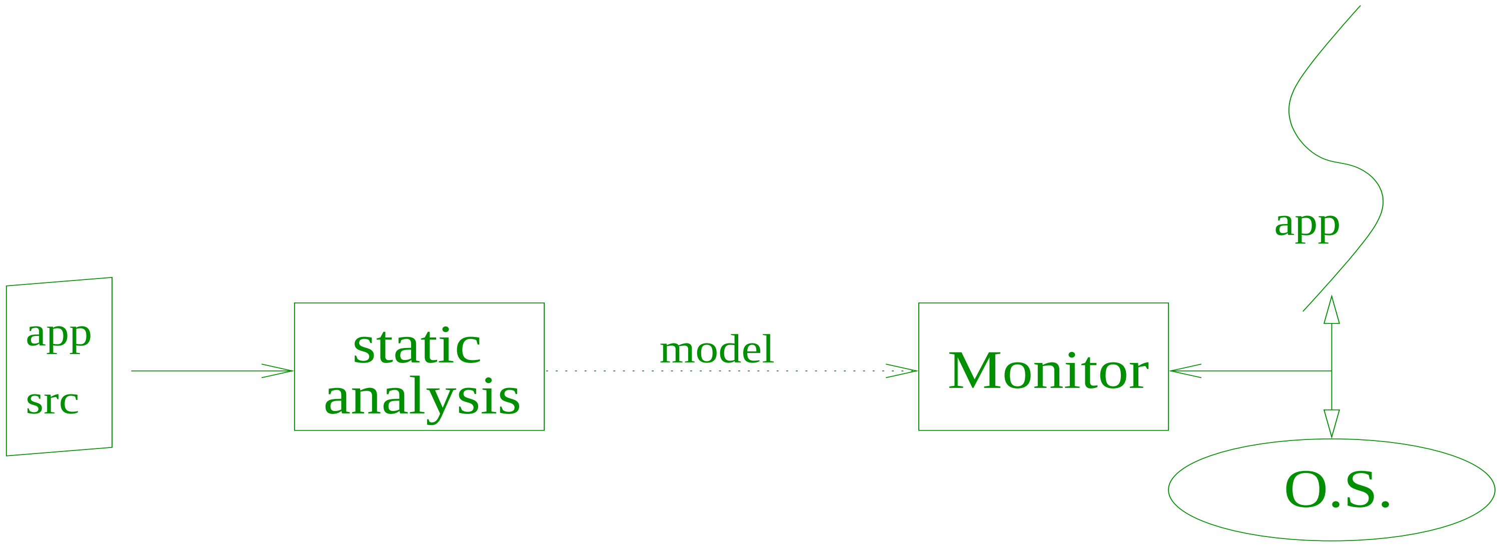
Static Analysis to The Rescue



A toy example:

- Model: The set S of system calls the program could ever call.
- Enforcement: If the program's syscall trace contains anything not in S, raise an alarm.

Benefits of Static Models

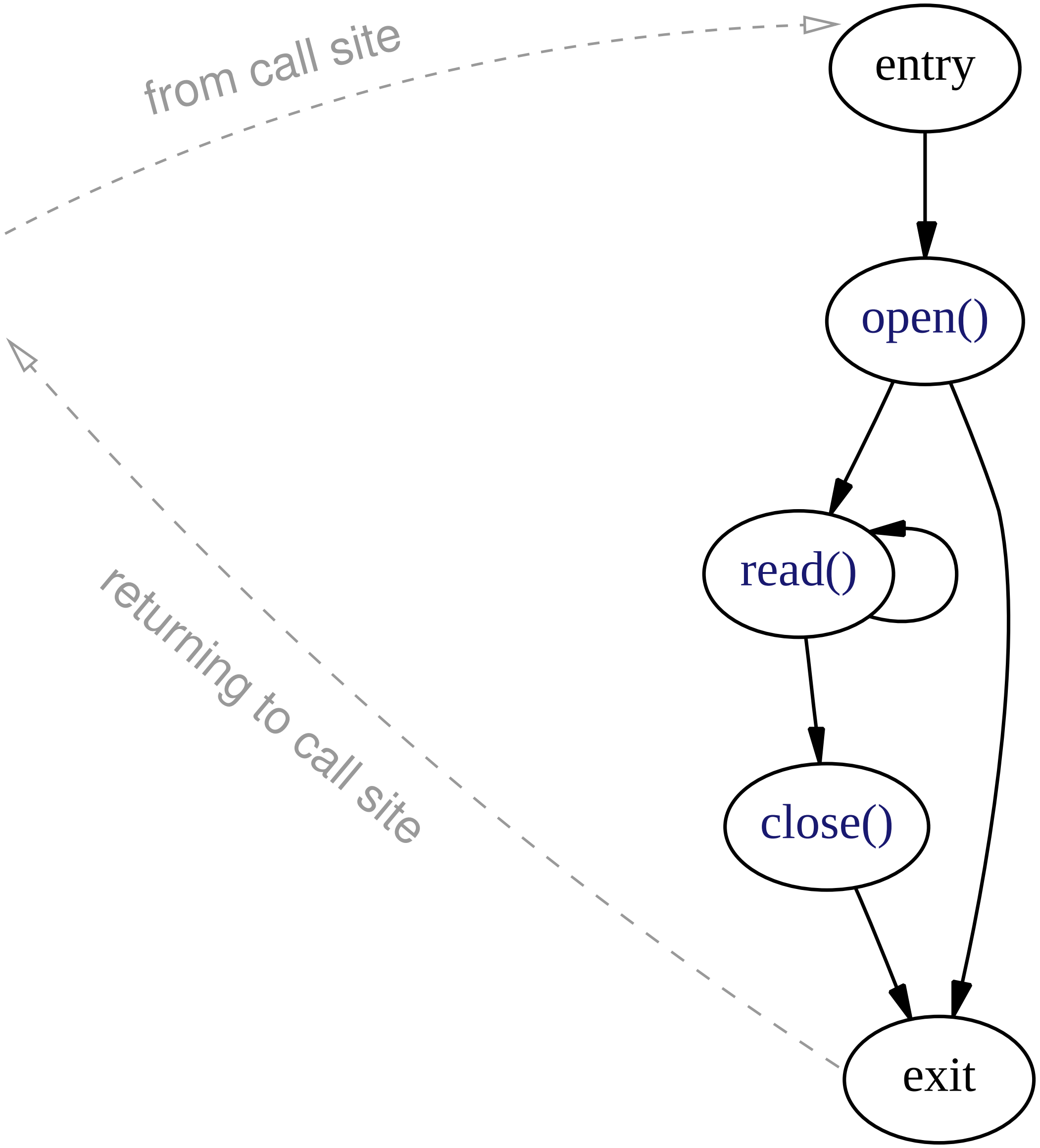


Advantages of static models:

- Model can be derived via *static analysis* of the program.
- Model is *complete*: it includes all correct behavior.

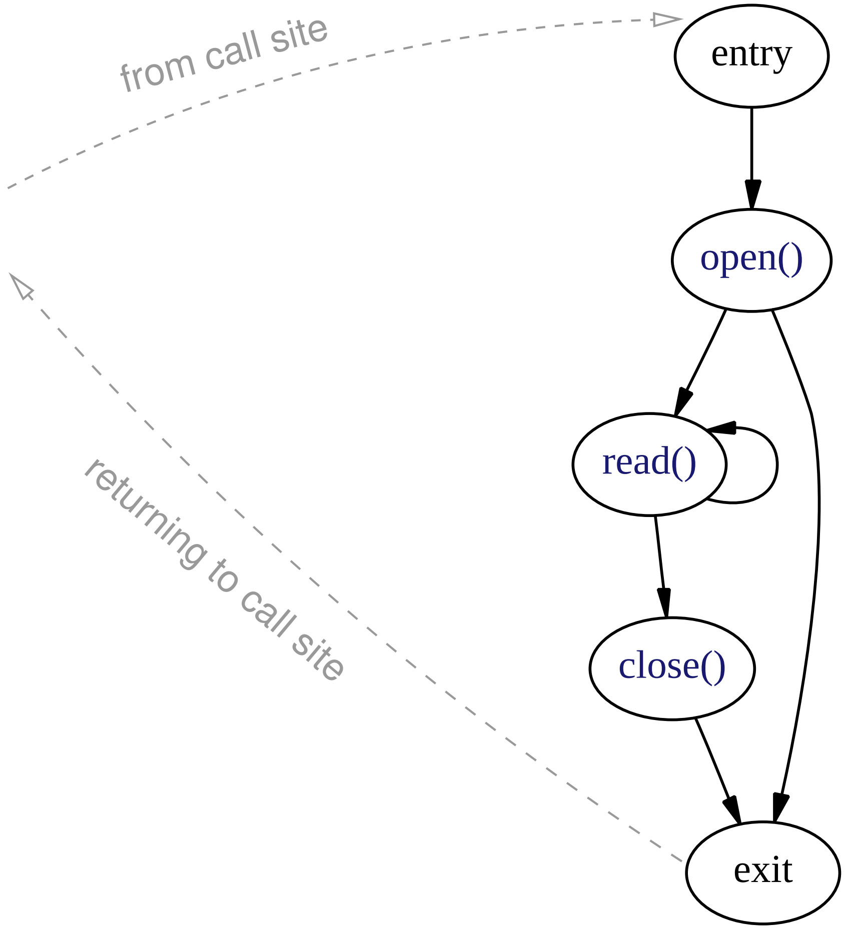
  ⇒ *No false alarms.*

A Control-Flow Graph



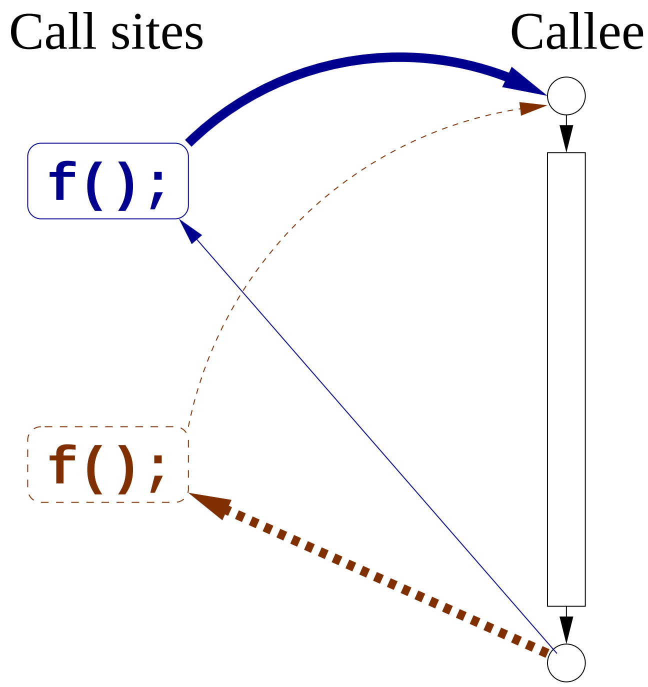
You might see this as an intermediate form in a compiler.

The Callgraph Approach



- Model: A finite automaton M.
- Enforcement: Does M accept the program's syscall trace?
- Model built from control-flow graph of the program.

The Abstract Stack Approach



A model that excludes impossible paths:

- Model: A non-deterministic pushdown automaton M.
- Enforcement: Can the program's syscall trace be *parsed* by M?

Implementing The Abstract Stack

Idea: use an *online* parser.

- First try: `yacc`
  - Doesn't work. Programs aren't $LR(1)$.
- Second try: breadth-first search
  - Doesn't scale. Big programs are too ambiguous.

    Readily goes exponential, even enters infinite loops!
- Third try: compress set of possible PDA-stacks
  - Following idea prevents exponential behavior:

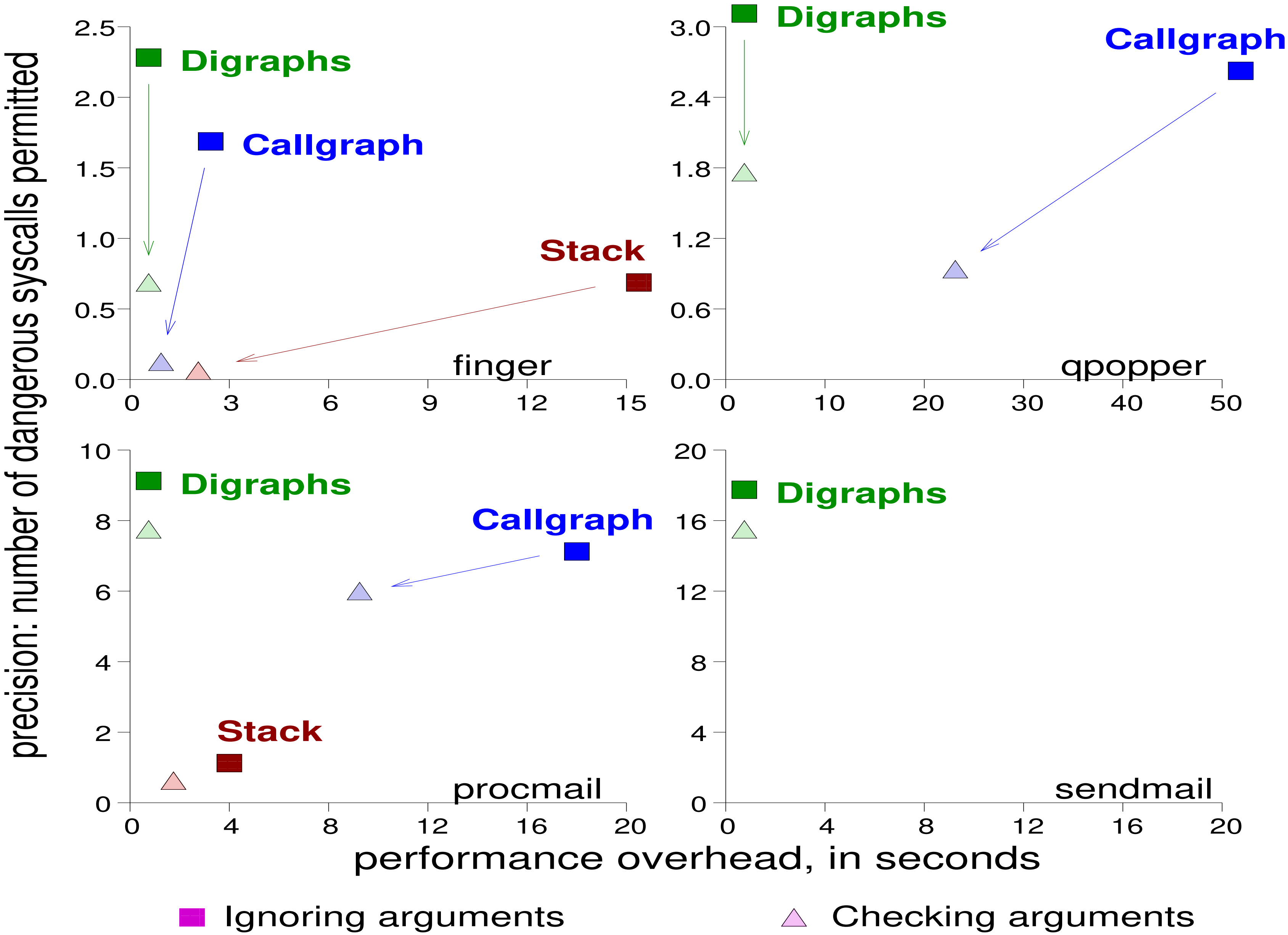
    Theorem 1. *Let S be the set of possible stacks of a PDA after reading some n input symbols. Then (1) S is a regular language, and (2) S can be recognized by a small NDFSA (with just $O(n)$ states).*

The Digraph Model

`open()` then `read()`
or
`read()` then `read()`
or
`read()` then `close()`
⋮

- Model: A set of pairs of system calls, $S \subseteq \Sigma^2$.
- Enforcement: All observed syscall pairs should be in S.
- Derivation: Let M be a NDPDA model; then $\langle x, y \rangle \in S$ iff $\Sigma^* xy \Sigma^* \cap L(M) \neq \emptyset$.

Our Implementation



The Importance of Unforeseen Attacks

- Many attacks exploit previously-unknown holes
  - Example: `telnetd` allowed attacker to control dynamic linker, execute arbitrary code on target machine

$\Rightarrow$ Not enough just to detect the attacks we already know about

Summary

- Key ideas:
  - New models for characterizing program behavior
  - Static analysis lets you constrain program behavior to be consistent with the source code
- Static analysis changes the way you build IDS's
  - Not a silver bullet, but . . .
  - *No false alarms.*