

# Challenges for Information-flow Security\*

Steve Zdancewic  
University of Pennsylvania

\* This talk is an attempt to be provocative and controversial.

# Protecting Confidential Data

---

---

- Protecting secret or private information in computer systems is a well known and long standing problem:
  - Modern OS's use access controls to protect files
  - Common practice to encrypt data sent on Internet
- But these mechanisms are insufficient:
  - Access controls don't prevent propagation of information
  - Can't feasibly compute over encrypted data

# Information-flow Security

---

- Alternative idea: tag “secret” information and track how it is used by the system.
  - An old idea (studied in computer system context since the 70’s, probably earlier)
- Dynamic enforcement
  - Literally tag the data
  - Conservatively propagate the tags
  - Expensive and to deal with implicit flows (need to know information about untaken paths)
- Static enforcement
  - Virtually “tag” the data (using types or some other annotations)
  - Program analyses determine whether program leaks information (does know all potential paths)

# Prior Research

---

---

- Sabelfeld and Myers' survey: "Language-based Information-flow Security" (2003)
  - More than 147 Papers related to information-flow security
  - Most concerned with definitions/refinements/variations on "noninterference"
- Despite this attention, and its appeal, information-flow security is not used!

Why?

# Non-problem #1

---

---

- Giving better, more precise definitions of noninterference-like properties for more complicated programming models.
  - Security is always defined relative to some level of abstraction and there are always attacks that violate the abstraction.
  - Any application that would benefit from a more complex/refined security analysis would also benefit from a coarser analysis.
  - We don't have any killer apps for existing information-flow technology.

# Non-problem #2

---

- Implementing a “practical” system for enforcing information-flow policies.
  - There exist two quite sophisticated, full-featured programming languages that provide information-flow enforcement.
- Flow Caml
  - François Pottier and Vincent Simonet
  - OCaml dialect with information-flow labels, label polymorphism, type inference, etc.
  - <http://cristal.inria.fr/~simonet/soft/flowcaml/>
- Jif
  - Andrew Myers and his students
  - Variant of Java with information-flow labels, dynamic principals, type inference, etc.
  - <http://www.cs.cornell.edu/jif/>

# Clarification

---

---

- This is not to say that there are no interesting research questions in basic information-flow theory and its implementation.
  - Examples: Concurrency and security-preserving compilation (among others)
- But: Theory and implementation aren't the critical barriers to putting the technology into practice.

So what *are* the challenges?

# Three (of many) Challenges\*

---

- Integrating with existing / other security infrastructure
- We don't want noninterference anyway
- Policy is *hard*

\* no answers... only questions

# Info-Flow Control Integration

---

---

- Problem: Combining info-flow controls with other kinds of security policies/mechanisms
- Other software:
  - Legacy code
  - Existing interfaces can't be easily changed
  - Wrappers? (will likely be conservative or unsound)
- Other mechanisms
  - File systems access controls, OS protections, PKI, authentication mechanisms, cryptography, audit logs, ...
  - Real programs need to do I/O

# Info-Flow Control Integration (2)

- There has been some work in this direction:
  - Chothia, Duggan, Vitek: “Type-based Distributed Access Control” CSFW 2003
  - Banerjee & Naumann: “Using Access Control for Secure Information Flow in a Java-like Language” CSFW 2003
  - Tse & Zdancewic: “Run-time Principals in Information-flow Type Systems” IEEE S&P 2004
- Example: Information-flow policy integrated with OS abstractions such as users and file permissions

# Integration Example

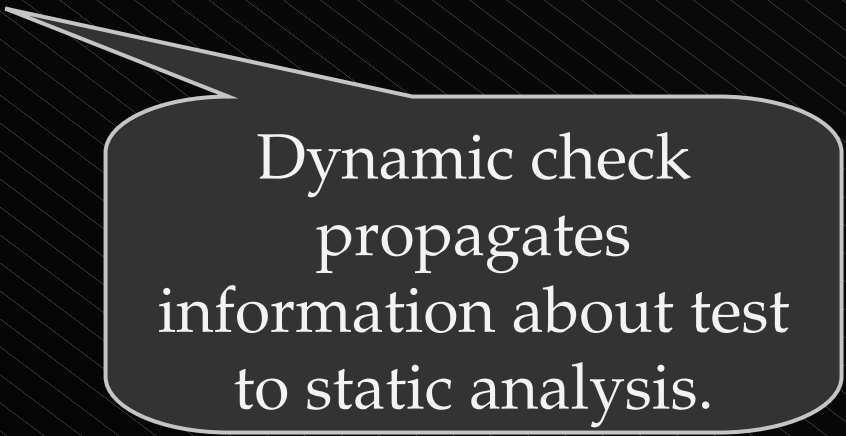
---

```
String{root:root} secret;

void System.print(String{any:System.user} s);

void printIfRoot(String{any:root} s) {
    if ActsFor(System.user, root) {
        System.print(s);
    }
}

printIfRoot(secret);
```



Dynamic check  
propagates  
information about test  
to static analysis.

# We Don't Want Noninterference

- Noninterference is an extremely restrictive policy
  - Most real systems *are* intended to leak some information
  - If the policy truly is noninterference, then least privileges argues for implementing separate systems (high and low)
  - Analyses are conservative, so even if the desired system is noninterfering, it does not necessarily pass the test
- So we need to escape the confines of noninterference

# Downgrading

---

- Intransitive noninterference
  - Ensures that downward information flows pass through trusted components
- Built in primitives
  - Encryption may justify declassification, so build encryption in as a primitive
  - Volpano & Smith's comparison operator from their work on "Relative Secrecy" (POPL 2000)
  - Not very general
- Add a "declassify" or "endorse" operator
  - How to constrain its use?
  - Jif's model: authority (need permission of those who could be harmed)
  - Robust downgrading (Myers, Sabelfeld, Zdancewic)
- Hard to reason about downgrading (See Andrei's talk...)

# Policy is Hard

---

---

- Realistic policies are complex
  - Many principals or stakeholders
  - Conditions under which downgrading may occur
  - Spectrum between static policies and policies that depend on dynamic information
- Even for small programs, there are many reasonable choices for the info.-flow policy
  - People have difficulty managing the small set of access control bits most file systems provide
  - Software engineering issue is highly nontrivial
  - When something goes wrong, it is often difficult to determine the source of the problem
  - Difficult to formalize intuitive policies using annotations

# Policy Engineering

---

---

- Some existing mechanisms can help:
  - Polymorphism
  - Inference
- Lack of experience with building real systems that enforce information-flow policies means we have no idea how to design real policies.

Where are the killer apps for information flow?

# Perl Taint Checking

---

---

- The Perl scripting language supports dynamic information-flow checking
  - Data is tagged as “tainted” or “untainted”
  - Propagates tags through basic operations
  - Has no support for preventing implicit flows (so it enforces “secrecy” as opposed to noninterference)
  - Provides a built in downgrading mechanism via pattern matching (but unsafe—not sufficient to protect against malicious code)
  - Has a built in policy: data from the network and user input is “tainted”; system calls require untainted data
  - Is widely deployed because it is believed to help prevent buffer overflow and format-string violation errors.
  - Not good for confidentiality?

# Lessons from Perl

---

---

- Simplicity is key
  - Default behavior is reasonable
  - But how general can a “fixed” confidentiality analysis be?
- Unsoundness is OK
  - Only trying to *improve* security not guarantee it against all attackers
- We should look for other interesting policies between access control and information-flow

# Challenge

---

---

- Information-flow security is seductive (and has been for > 30 yrs!) but has not been very useful in practice
- We need to demonstrate the utility of information-flow approaches
  - Interface to other system components
  - Noninterference is not the right policy
  - Figuring out the the right policies is hard
- Download Flow Caml or Jif and build something with them!