

The logo for TIME DC, featuring the words "TIME" and "DC" in a stylized, orange, outlined font against a blue gradient background.

Trustworthy Infrastructure, Mechanisms and Experimentation
for Diffuse Computing

Steve Zdancewic

Ph. D. students:

Peng Li

Stephen Tse



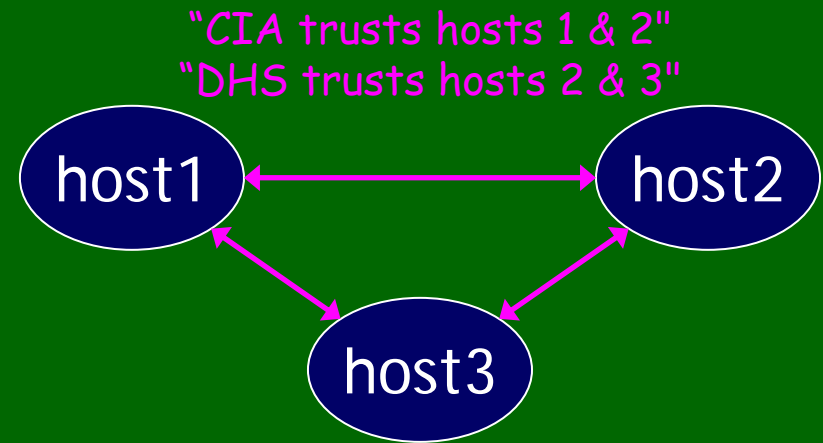
Security-Oriented Languages

University of Pennsylvania

TIME DC Objectives

- Efficient and secure distributed information systems:

- Allow network defenders to exchange information
- Military & government communications
- Businesses that rely on computing infrastructure



- Technical challenges:

- Software is large, distributed, and complex
- Security policies become complex (especially when multiple parties are concerned)
- Existing mechanisms (e.g. OS, crypto, malware defenses) are crucial... need to integrate with them

Secure diffuse systems?

- Need: distributed systems that provide information security—confidentiality, integrity, anonymity
 - Problem 1: controlling information release
 - Problem 2: (mutual) distrust
- TIME DC researchers looking at many aspects of this problem:
 - Scedrov & Mitchell : formal protocol and policy analysis
 - Feigenbaum: incentive-based multiparty collaboration
 - Halpern: reasoning about policies
- How SOL (Security-oriented Languages) fits in:
 - Addresses software aspects, particularly heterogeneous trust and protecting confidential information
 - Connect policy to implementation
 - Slogan “Building programs that are *secure by construction*”

Language-based Security

● Static Analysis

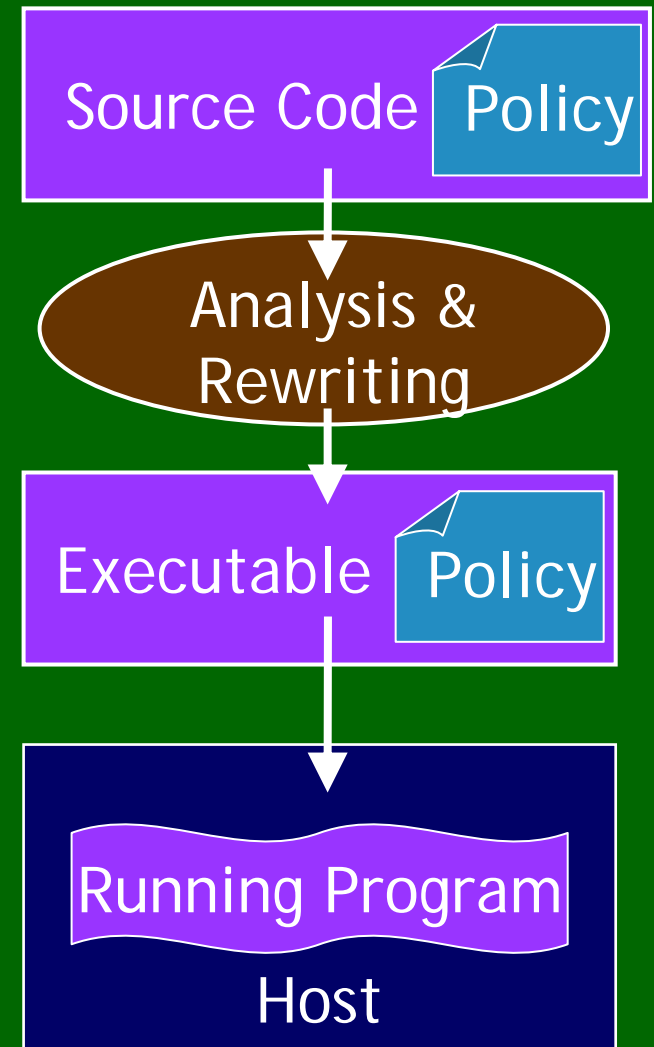
- Type Systems [Java, C#, ML,...]
- Proof Carrying Code [Necula / Morissett]

● Program rewriting

- Code instrumentation
- Inlined reference monitors [Schneider et al.]

● Dynamic monitoring

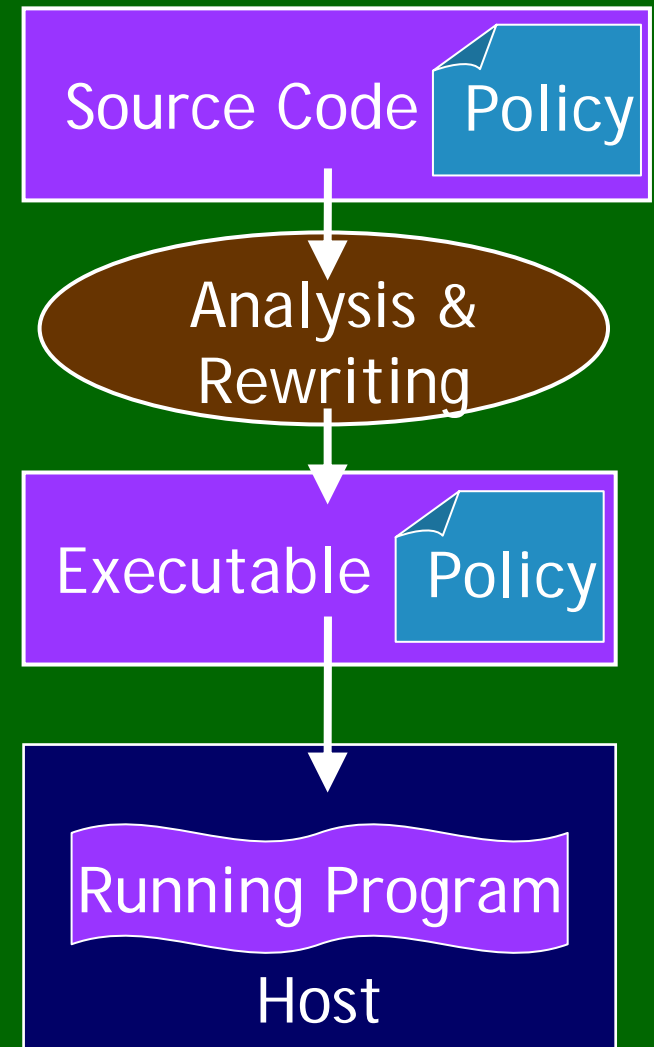
- Java/C# stack inspection
- Access Control Checks



Language-based Security

Benefits:

- Automated tools
- Explicit, fine-grained policies
- Program abstractions
- Regulate end-to-end behavior
- Increased confidence in security of the system



Information Security Policies

[Myers & Liskov 2000]

- Confidentiality labels:

`int{Alice:} a1;` "a1 is Alice's private int"

- Integrity labels:

`int{*:Alice} a2;` "Alice trusts a2"

- Combined labels:

`int{Alice: ; *:Alice} a3;` (Both)

Enforced by a compiler using static information flow analysis:

```
int{Alice:} a1, a2;  
int{Bob:} b;  
int{*:Alice} c;
```

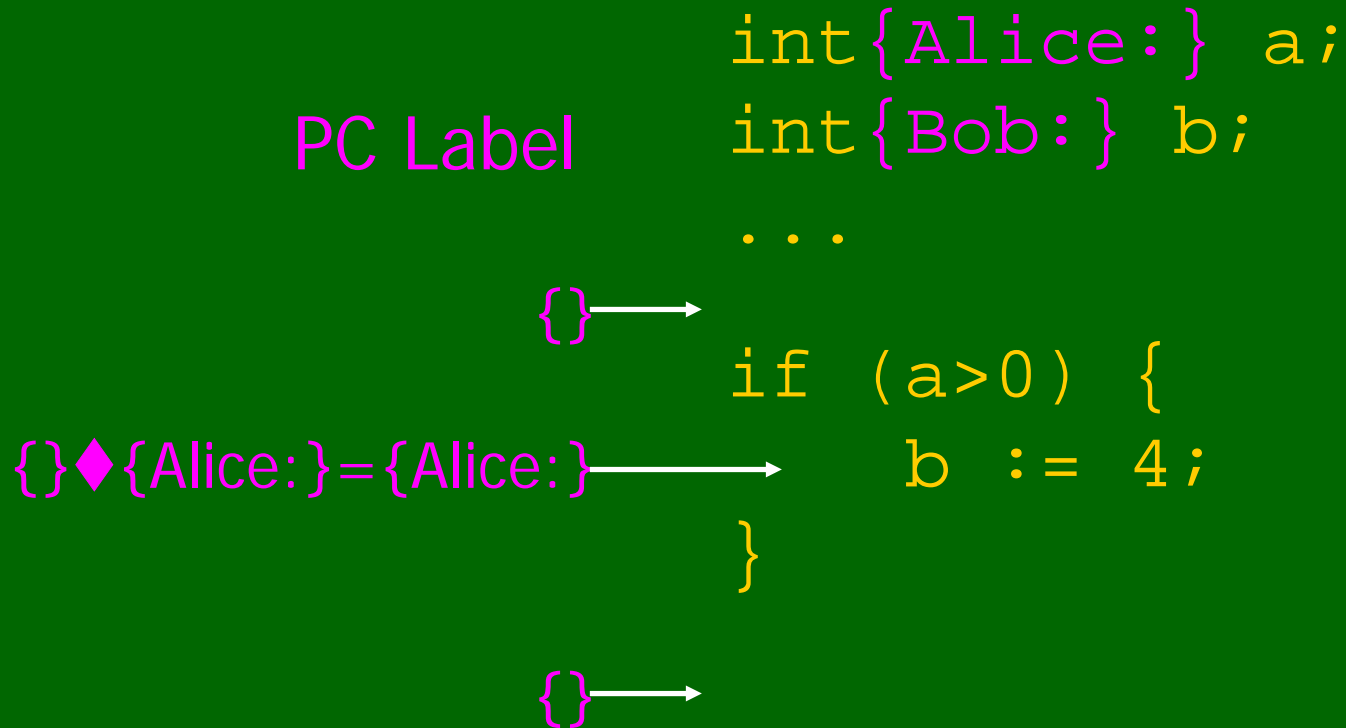
Insecure

```
a1 = b;  
b = a1;  
c = a1;
```

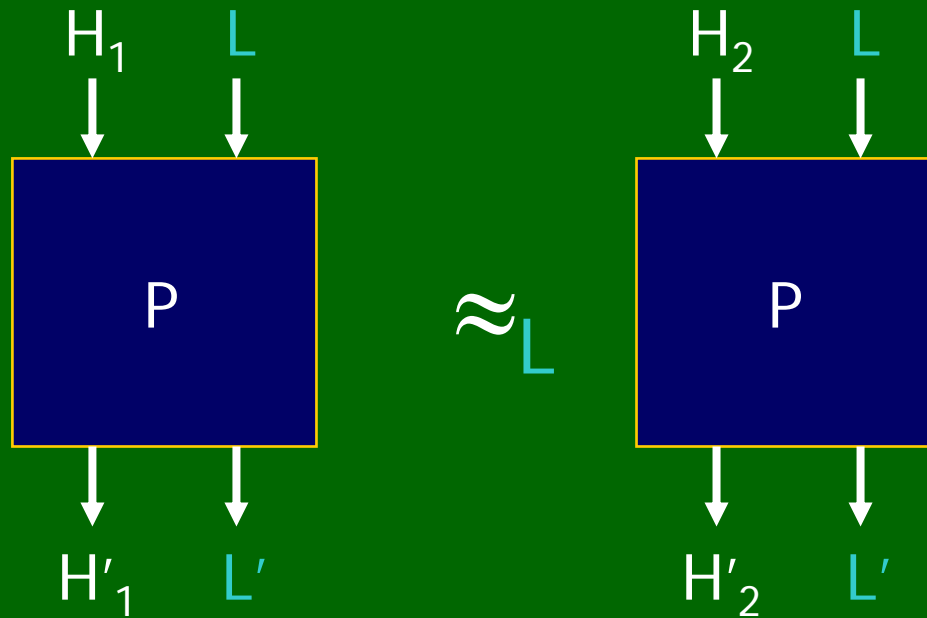
Secure

```
a1 = a2;  
a1 = c;
```

Simple State & Implicit Flows



Noninterference



- Proved by:
 - Logical relations
 - Bisimulation techniques
- But insufficient for practical software

Dynamic Policies

- Security policies that involve run-time information:

```
principal user = login();  
String{root:} secret = ...;
```

```
/* printed information is visible to user */  
void print (String{user:} x) {...}
```

```
/* Shouldn't print root's data unless user is root */  
void printroot (String{root:} y) {  
    if (user == root) print(y);  
}
```

More complex policies

- Downgrading:

"Alice will release her sensitive data to Bob, but only after he has paid her for it."

```
int{Alice:} s = ...;
if (hasPaid()) {
  y = declassify (s, {Alice:Bob} to {*:Alice});
}
```

- Authority:

- Restricts the program scopes in which declassify may be used

- Robust downgrading: integrity constraint

- A principal whose security policy is weakened must trust the integrity of the decision to do the downgrading.

[ZM'01, Zdd'03, SMZ'04]

Connection to PKI

- Runtime principals means that there must be a runtime representation of authority:
 - Witness of authority for uses of declassification on data owned by dynamic principals
- Digital certificates
 - Identify principals and public keys
 - Authority represented by digital certificates
- Programming language point of view:
 - Add a new type of data "cert"
 - Add a way to abstractly verify these certificates

ATM/Banking Example

```
void main (cert cnet) {
  principal{ATM:} id = login();

  cert cdel = acquire id :> Delegate ATM;
  cert creq = acquire id :> Withdraw 100;

  Msg{ATM:} m = new Msg(id, cdel, creq);
  if (cnet => ATM :> Declassify Net) {
    Msg{ATM:Net} x = declassify m as Msg{ATM:Net};
    Int{user:} balance = request(x);
    ...
  }
  ...
}
```

Run-time Principals in Information-flow Type Systems

- Joint work with Stephen Tse
- Published in IEEE Security & Privacy 2004
- Formalizes these dynamic policies
 - Proves a soundness result for the compiler analysis
- Demonstrates how dynamic policies are related to public-key infrastructure
 - Brings together language-based research and existing work on PKI

Downgrading Policies and Relaxed Noninterference

- Joint work with Peng Li
- To be published at conference on Principles of Programming Languages (POPL) 2005
- Extends declassification operation to describe the circumstances under which declassification is permitted
 - The paper gives a semantic characterization of the impact of such declassification

Ongoing & Future Work

- **Implementation**

- Jif - Java + information flow (based at Cornell)
- Apollo - a lightweight prototype (based at Penn)

- **Web-scripting languages**

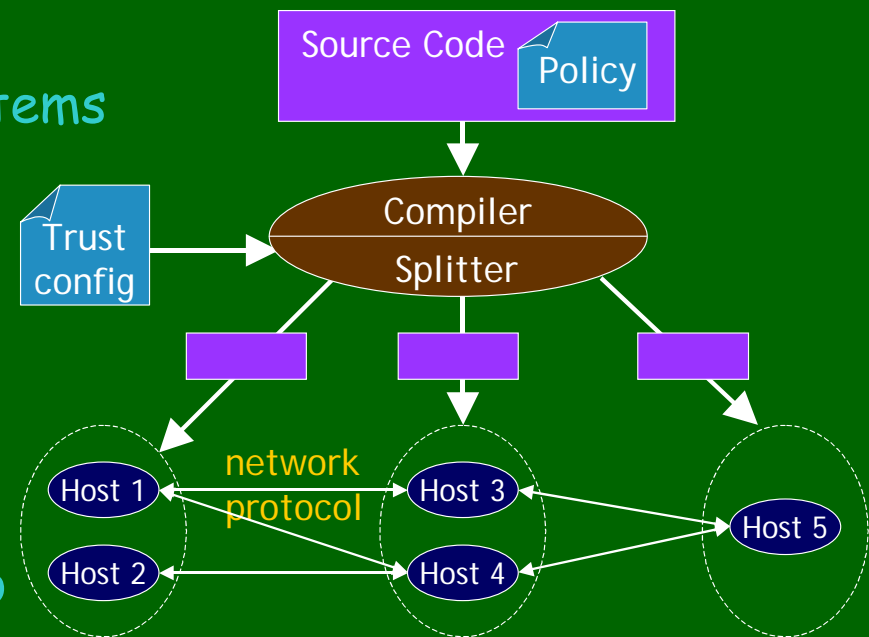
- Web-based interface to information distribution systems
- Challenges with aggregate information

- **Cryptographic protocol implementation**

- Connect specification as in Scedrov & Mitchell's work to the implemented system

- **Incorporate richer policy languages**

- Halpern's work on logics for policies



Security-oriented Languages:

- Looking at the software implementation aspects of building secure diffuse computing infrastructures.
- Still have many interesting open problems.
- This collaboration is just getting started...

Recent Papers:

<http://www.cis.upenn.edu/~stevez/sol/>

A Design for a Security-typed Language with Certificate-based Declassification Downgrading Policies and Relaxed Noninterference - POPL 2005
Advanced Control Flow in Java Card Programming - LCTES 2004
Translating Dependency into Parametricity - ICFP 2004
Enforcing Robust Declassification - CSFW 2004
Run-time Principals in Information-flow Type Systems - Security & Privacy 2004
Information Integrity Policies - FAST 2003