Language-based Information-flow Security

Steve Zdancewic

University of Pennsylvania

Confidential Data

- Networked information systems:
  - PCs store passwords, e-mail, finances,...
  - Businesses rely on computing infrastructure
  - Military & government communications
- Security of data and infrastructure is critical [Trust in Cyberspace, Schneider et al. '99]
- How to protect confidential data?

Don’t take my word for it…

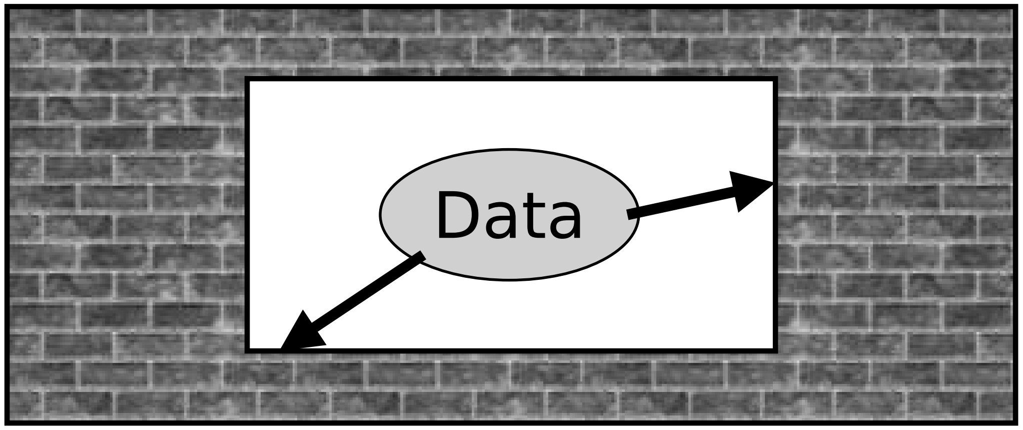
“Users should be in control of how their data is used. Policies for information use should be clear to the user. Users should be in control of when and if they receive information to make best use of their time. It should be easy for users to specify appropriate use of their information including controlling the use of email they send.”

--Bill Gates, January 15, 2002

Technical Challenges

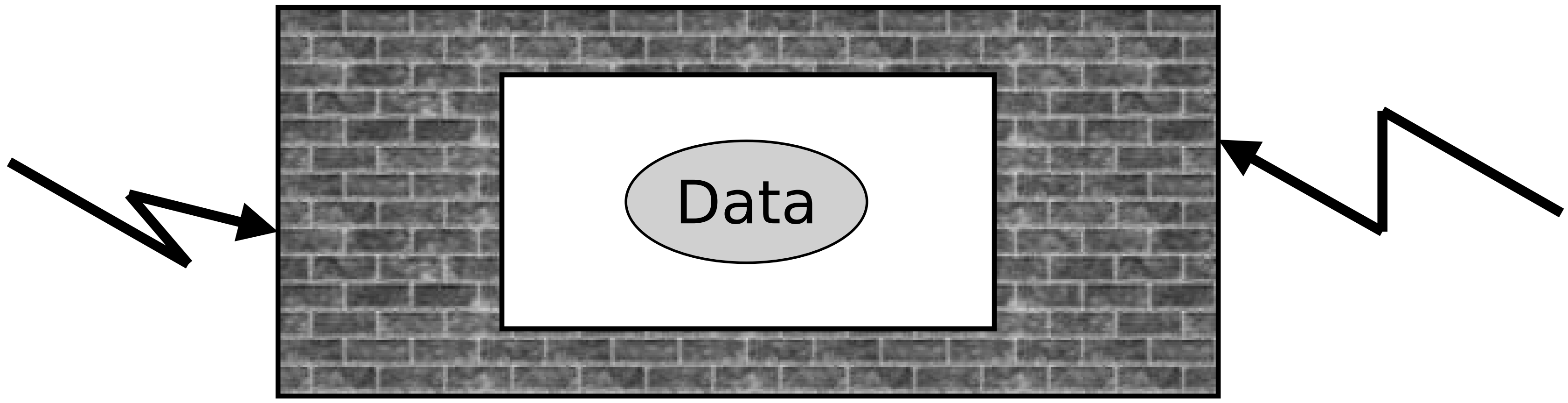
- Software is large and complex
  - Famous bugs: e.g. MS HotMail
  - Buffer overflows
- Security policies are complex
  - Mutually distrusting parties
- Requires tools & automation
- Look at traditional security concerns to set the context…
  - Confidentiality
  - Integrity
  - Availability

Quality 1: *Confidentiality*



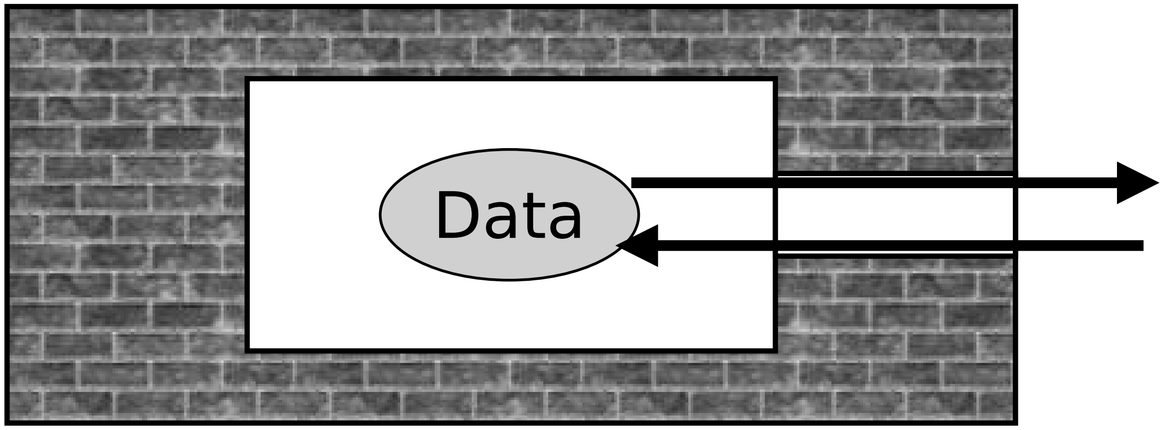
- Keep data or actions *secret.*
- Related to: Privacy, Anonymity, Secrecy
- Examples:
  - Pepsi secret formula
  - Medical information
  - Personal records (e.g. credit card information)
  - Military secrets

Quality 2: *Integrity*



- Protect the *reliability* of data against unauthorized tampering
- Related to: Corruption, Forgery, Consistency
- Example:
  - Bank statement agrees with ATM transactions
  - The mail you send is what arrives

Quality 3: *Availability*



- Resources usable in timely fashion by authorized principals
- Related to: Reliability, Fault Tolerance, Denial of Service
- Example:
  - You want the web-server to reply to your requests
  - The military communication devices must work

Access Control

- Access control
  - e.g. File permissions
  - Access control lists or capabilities
  - Modern spin: Stack inspection
- Drawback:
  - Does not regulate propagation of information after permission has been granted.

Cryptography

- Essential for:
  - Protecting confidentiality & integrity of data transmitted via untrusted media
  - Authentication protocols
- Drawbacks:
  - Impractical to compute with encrypted data!
    - There are secret sharing techniques.
  - Doesn’t prevent information propagation once decrypted

End-to-end Solution

- Rely on access control & encryption
  - Essential (authentication, untrusted networks, etc.)
- But… also use language techniques:
  - verify programs to validate information flows that they contain.

Benefits **(of PL-based mechanisms)**

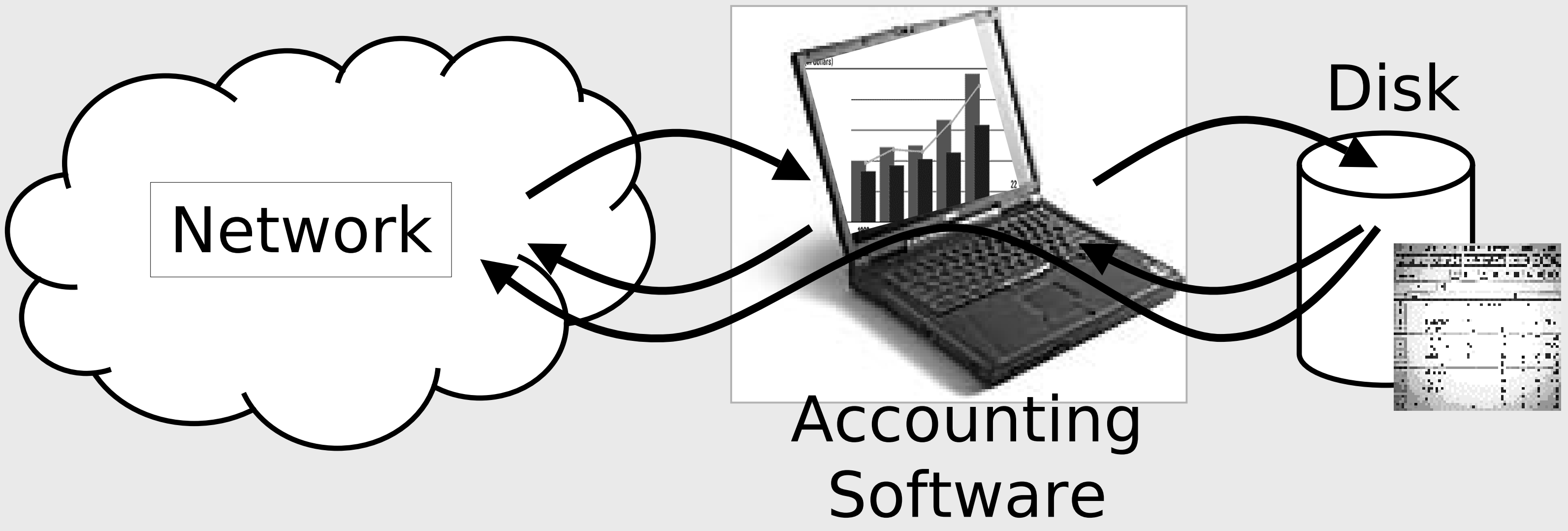
- Explicit, fine-grained policies
  - Level of single variable if necessary
  - TAL/PCC level
- Program abstractions
  - Programmers can design custom policies
- Regulate end-to-end behavior
  - Information Flow vs. Access Control
- Tools: increase confidence in security

Focus of These Lectures

- Confidentiality (& weak integrity)
- How to define information security?
- How to enforce it?
  - Type Systems for information-flow security
  - Proof of security
- Scaling it up
  - Polymorphism
  - Datatypes
  - State & Effects
- Challenges & practicality
  - Decentralized label model (Jif)
  - Downgrading (declassification)

Information-flow Policy

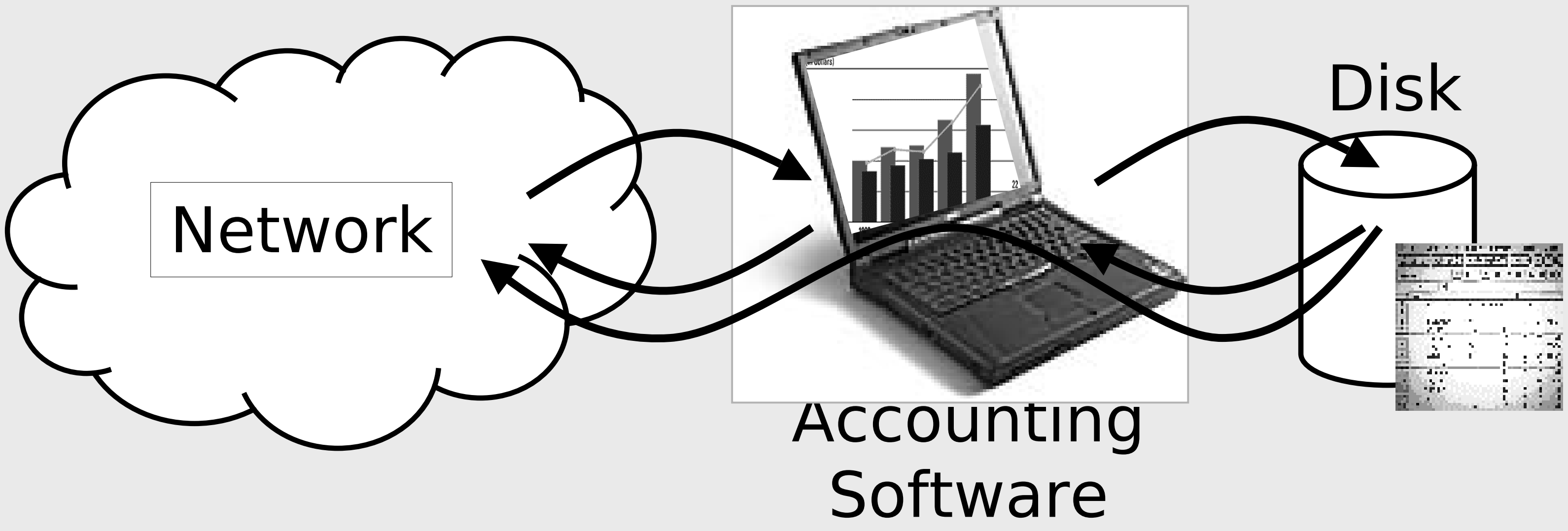
- Downloadable financial planner:



- Access control insufficient
- Encryption necessary,but not end-to-end

Noninterference

[Reynolds ’78, Goguen&Meseguer ’82,’84]

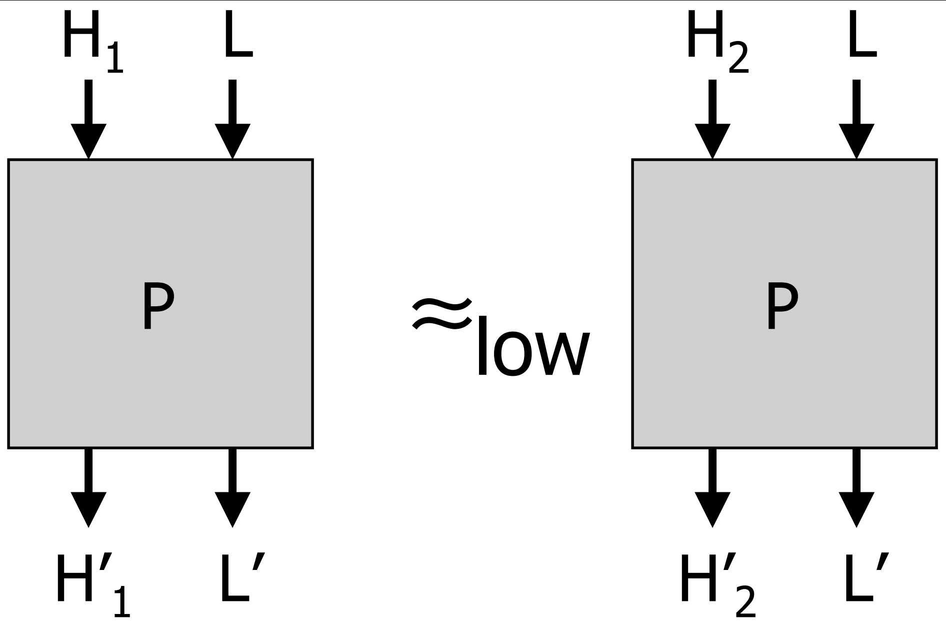


- Private data does not *interfere* with network communication
- Baseline confidentiality policy

Comparison to Secrecy in Spi

- Spi considers secrecy of atomic keys
  - Keys can be manipulated in limited ways (i.e. encryption & decryption)
  - Cryptographic primitives are assumed to be perfect (or probabilistically secure)
  - Not possible to leak partial information
- Contrast to arbitrary datatypes
  - Can be manipulated in many ways
  - Possible to leak partial information

Noninterference



- Proved by:
  - Logical relations
  - Bisimulation techniques

Formalizing Noninterference

- Original formulation: Trace-based models of computation
  - Goguen & Meseguer 1982
  - McClean – late 1980's early 1990's

- Dorothy Denning proposed program analysis techniques
  - Mid-late 1970's (but no proofs of correctness)
- Experiments with Multics
- Volpano & Smith 1996
  - Type system for noninterference

- See Sabelfeld & Myers 2003 for survey.

External Observation

- *External behavior*
  - Observations seen by someone “outside” the system
  - Outputs (i.e. strings printed to terminal)
  - Running time
  - Power or memory consumption
  - Comments
  - Variable names
- Very hard to regulate!
  - There is always some attack below the level of abstraction you choose.
  - But… attacks against external behavior tend to be difficult to carry out and/or have low bandwidth

Internal Observation

- *Internal behavior*
  - At the programming language level of abstraction
  - Note that many “external observations” can be internalized by enriching the language (e.g. add a clock)
- Observational equivalence
  - $e_1 \approx e_2$ iff for all C[]. $C[e_1] = C[e_2]$
  - $C[e_1] \rightarrow^* v$ iff $C[e_2] \rightarrow^* v$

Observations

- Final output of the program.
  - Pure lambda calculus
- Evaluation order
  - Lambda calculus with state
- Thread scheduling decisions
  - Multithreaded languages with state/message passing

Low Equivalence

- Captures what a “low-security” observer can “see”
- Example: Suppose program states consist of pairs: high * low

`(“attack at dawn”, 3)` $\approx_{low}$ `(“stay put”, 3)`

`(“attack at dawn”, 3)` $\not\approx_{low}$ `(“stay put”, 4)`

Lattice Model of Policies

- Proposed by Denning '76
- Use a lattice of *security labels*
  - Higher in lattice is more "confidential" or "secret"
  - Use $\sqsubseteq$ for order relation
  - Use $\sqcup$ for join (l.u.b.)
  - Use $\sqcap$ for meet (g.l.b.)
- Prototypical example: low $\sqsubseteq$ high

Simply-typed secure language

```
t ::= bool  |  s → s      types
s ::= t{l}                secure types

v ::= x | true | false    values
   |  λx:s.e

e ::= v                   values
   | (e e)                application
   | e ⊕ e                primitive op.
   | if e then e else e   conditional
```

Semantics

- Large step operational semantics

- Static semantics
  - Lattice lifted to a subtyping relation
  - “Standard” information-flow type system
  - Heintze & Riecke’s SLam calculus POPL’98
  - Pottier & Conchon ICFP’00

Noninterference Theorem

If $x{:}t\{hi\} \vdash e : t'\{low\}$

$\vdash v_1, v_2 : t\ \{hi\}$

$hi \not\sqsubseteq low$

then

$$e\{v_1/x\} \Downarrow v$$

iff

$$e\{v_2/x\} \Downarrow v$$

Proof

- Uses a logical relations argument
- Two terms are related at a security level L if they “look the same” to observer L

- Define logical relations
- Subtyping lemma
- Substitution lemma

Scaling Up

- Polymorphism & Inference
- Sums
- State and effects
    - Simple state
    - References
- Termination & Timing

Polymorphism & Inference

- Add quantification over security levels
  - $\forall$L::label. (`bool{L}` $\rightarrow$ `bool{L}){L}`
  - Reuse code at multiple security levels.
- Inference of security labels
  - Type system generates a set of lattice inequalities
  - Equations have the form $l \sqsubseteq l_1 \sqcup \ldots \sqcup l_2$
  - Constraint of this form can be solved efficiently

Polymorphism in Flow Caml

- Lists in Flow Caml
  [Vincent Simonet & François Pottier ’02,’03]
- Base types parameterized by security level bool{low} = low bool
- Type of lists also parameterized:
  ∀’a::label. ∀’L::type. (‘a, ‘L) list

```
x1 : hi int
[1;2;3;4] : ('L int, 'M) list
[x1; x1]  : (hi int, 'L) list
```

Example: List Length

- Length does not depend on contents of list:

```
let rec length = function
    [] -> 0
  | _ :: tl -> 1 + length tl
:
 (‘a, 'M) list -> 'M int
```

Example: has0

- Lookup depends on both contents and structure of the list:

```
let rec has0 = function
    [] -> false
  | hd :: tl -> hd = 0 || has0 tl
:
  ('L int, 'L) list -> 'L bool
```

Sums & Datatypes

- In general: destructors reveal information
- Accuracy of information-flow analysis is important [Vincent Simonet ’02]
- Suppose x:bool{L_1}, y:bool{L_2}, z:bool{L_3}

```
datatype t = A | B | C
let v = if x then (if y then A else B)
             else (if z then A else C)
let i = case v of
  A | B -> 1
    | C -> 0
```

- What is label of i?

Simple State & Implicit Flows

```
                              int{high} a;
          PC Label            int{low} b;
                              ...
             {low} ⟶
                              if (a>0) {
{low}⊔{high}={low} ⟶              b := 4;
                              }

             {low} ⟶
```

Simple State & Implicit Flows

```
                                   int{high} a;
           PC Label                int{low} b;
                                   ...
                 {low} ⟶
                                   if (a>0) {
 {low}⊔{high}={high} ⟶                b := 4;
                                   }
```

To assign to variable with label L, must have $PC \sqsubseteq L$.

Full References: Aliasing

```
h:int{high}

let lr = ref 3 in
let hr = lr in
  hr := h
```

Information leaks through aliasing:
Both the pointer *and* data pointed to can cause leaks.

Two more leaks

```
h:int{high}

let lr1 = ref 3 in
let lr2 = ref 4 in
let lr' = if h then lr1 else lr2 in
  l := !lr'

let lr1 = ref 3 in
let lr2 = ref 4 in
let lr' = if h then lr1 else lr2 in
  lr' := 2
```

Secure References

```
t ::= … |  s ref                types
s ::= t{l}                      secure types

v ::= … | r                     heap pointers



e ::= …
   |  ref e                     reference alloc.
   |  !e                        dereference
   |  e := e                    assignment
```

Type System for State

- Modified type system for *effects*

  [Jouvelot & Gifford '91]
- pc label approximates control-flow info.

$$\Gamma\ [pc] \vdash e : s$$

- Notation: lblof(t{L}) = L
- Invariant of this type system:

$$\Gamma\ [pc] \vdash e : s \quad \Rightarrow \quad pc \sqsubseteq \mathrm{lblof}(s)$$

Typing Rules for State (1)

$$\Gamma\ [pc] \vdash \mathrm{true} : \mathrm{bool}\{pc\}$$

$$\frac{\Gamma\ [pc] \vdash e : \mathrm{bool}\{L\} \qquad \Gamma\ [pc \sqcup L] \vdash e_1, e_2 : s}{\Gamma\ [pc] \vdash \mathrm{if}\ e\ \mathrm{then}\ e_1\ \mathrm{else}\ e_2 : s}$$

Typing Rules for State (2)

- Prevent information leaks through assignment.
- Recall that pc $\sqsubseteq$ L

$$\frac{\Gamma\,[pc] \vdash e_1 : s\ \mathrm{ref}\{L\} \qquad \Gamma\,[pc] \vdash e_2 : s \qquad L \sqsubseteq \mathrm{lblof}(s)}{\Gamma\,[pc] \vdash e_1 := e_2 \ :\ \mathrm{unit}\{pc\}}$$

Typing Rules for State (3)

$$\frac{\Gamma\,[pc] \vdash e : s\ \mathrm{ref}\{L\}}{\Gamma\,[pc] \vdash\ !e\ :\ s \sqcup L}$$

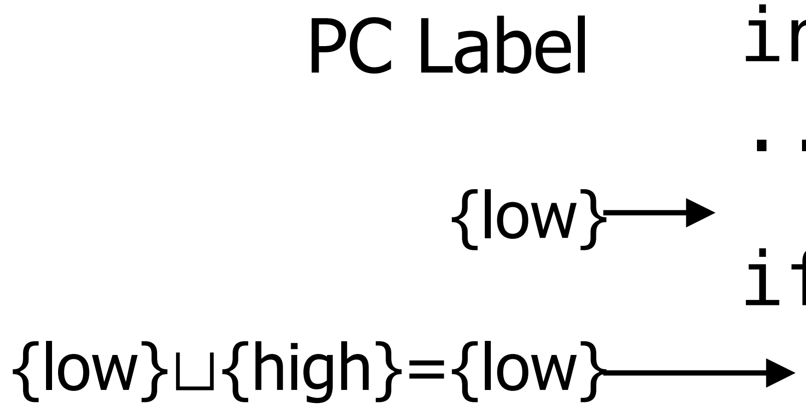
$$\frac{\Gamma\,[pc] \vdash e : s}{\Gamma\,[pc] \vdash \mathrm{ref}\ e\ :\ s\ \mathrm{ref}\{pc\}}$$

Function Calls

```
                                int{high} a;
             PC Label           int{low} b;
                                ...
                   {low} ──►
                                if (a>0) {
 {low}⊔{high}={high} ──────►        f(4);
                                }

                   {low} ──►
```

Function Calls



```
int{high} a;
int{low} b;
...

if (a>0) {
    f(4);
}
```

To call a function with effects bounded by L must have PC $\sqsubseteq$ L.

Effect Types for Functions

```
t ::= …  |  [pc]s → s    types
```

$$\frac{\Gamma, x{:}s_1\ [pc'] \vdash e : s_2}{\Gamma\ [pc] \vdash \lambda x{:}s_1.e : ([pc']s_1 \to s_2)\{pc\}}$$

Typing Application

$$\frac{\Gamma\,[\mathrm{pc}] \vdash e_2 : s_1 \qquad L \sqsubseteq \mathrm{pc}' \qquad \Gamma\,[\mathrm{pc}] \vdash e_1 : ([\mathrm{pc}']s_1 \to s_2)\{L\}}{\Gamma\,[\mathrm{pc}] \vdash e_1\ e_2 : s_2 \sqcup L}$$

More Effects

- Exceptions
  - Very important to track accurately
  - Related to sums

- Termination & Timing
  - Is termination observable?
  - For practicality sometimes want to allow termination channels.
  - Timing behavior can be regulated by padding (but is expensive!)

[Agat'00]

Practicality

- Expressiveness
- Full implementations: Flow Caml & Jif

- Decentralized label model
- Downgrading & Declassification

Expressiveness

- Languages are still Turing complete
  - Just program at one level of security
- How to formalize expressiveness?
- … I don’t know! (Try to write programs…)
- Agat & Sands ’01:
  - Considered strong noninterference with timing constraints
  - Algorithms take worst-case running time
  - Heapsort more efficient than quicksort!
  - Relax to probabilistic noninterference to allow use of randomized algorithms

Jif

- Jif – Java + Information Flow
  - Andrew Myers, Lantian Zheng, Steve Chong at Cornell
- Goal: Put this stuff into practice (Java)
- First step: enrich the policy language
- Principals: users, groups, etc.
  - Express constraints on data usage
  - Distinct from hosts
  - Alice, Bob, etc. are principals

Decentralized Labels

[Myers & Liskov '97, '00]

- Simple Component {owner: readers}
  - {Alice: Bob, Eve}

    "Alice owns this data and she permits Bob & Eve to read it."

- Compound Labels
  - {Alice: Charles; Bob: Charles}

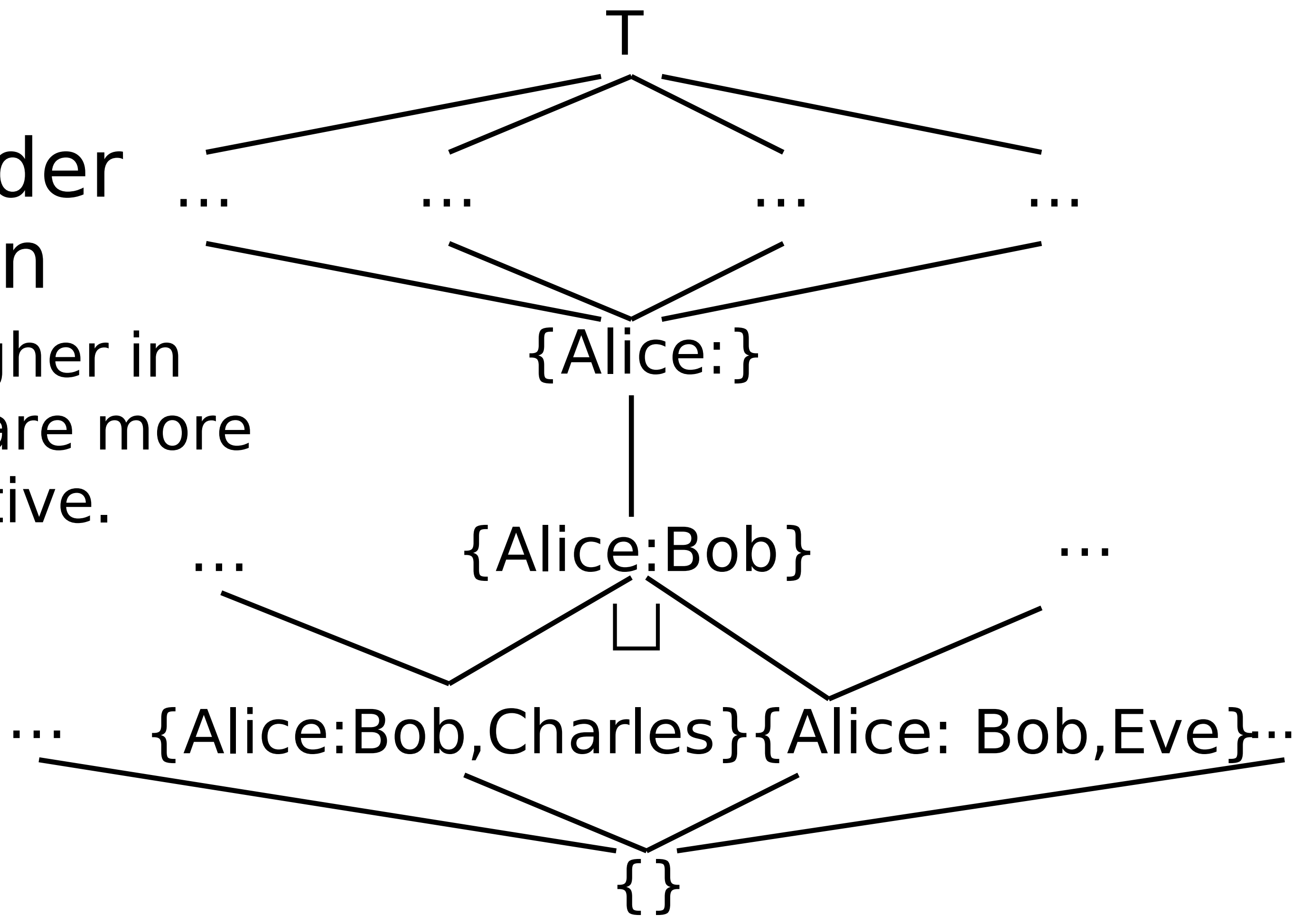
    "Alice & Bob own this data but only Charles can read it."

Decentralized Label Lattice

⊑ Order

⊔ Join

Labels higher in the lattice are more restrictive.



Integrity Constraints

- Specify who can write to a piece of data
  - {Alice? Bob}

"Alice owns this data and
she permits Bob to change it."

- Both kinds of constraints
  - {Alice: Bob; Alice?}

Integrity/Confidentiality Duality

- Confidentiality policies constrain where data can flow *to*.
- Integrity policies constrain where data can flow *from*.

- Confidentiality: Public $\sqsubseteq$ Secret
- Integrity: Untainted $\sqsubseteq$ Tainted

Weak Integrity

- Integrity, if treated dually to confidentiality is *weak*.
  - Guarantee about the source of the data
  - No guarantee about the quality of the data

- In practice, probably want stronger policies on data:
  - Data satisfies an invariant
  - Data only modified in appropriate ways by permitted principals

Richer Security Policies

- More complex policies:

"Alice will release her data to Bob, but only after he has paid $10."

- Noninterference too restrictive
  - In practice programs do leak some information
  - Rate of info. leakage too slow to matter
  - Justification lies outside the model (i.e. cryptography)

Declassification

```
int{Alice:} a;
int Paid;
...  // compute Paid
if (Paid==10) {
  int{Alice:Bob} b =
    declassify(a, {Alice:Bob});
  ...
}
```

“down-cast"
int{Alice:} to
int{Alice:Bob}

Declassification Problem

- Declassification is necessary & useful
- ...but, it breaks the noninterference theorem
  - Like a downcast mechanism
- So, must constrain its use. How?
  - Arbitrary specifications too hard to check.
  - Exploit the structure in the decentralized label model?

Robust Declassification

```
int{Alice:} a;
int{Alice?} Paid;
... // compute Paid
if (Paid==10) {
  int{Alice:Bob} b =
    declassify(a, {Alice:Bob});
  ...
}
```

Alice needs to trust the contents of paid.

Introduces constraint
PC $\sqsubseteq$ {Alice?}

[Zdancewic & Myers'01,Zdancewic'03]

Typing Rule for Declassify

$$\frac{\Gamma\ [pc] \vdash e : t\{L'\} \qquad PC \sqsubseteq \mathrm{auth}(L',L)}{\Gamma\ [pc] \vdash \mathrm{declassify}(e,\{L\}) : t\{L\}}$$

auth(L’,L) - returns integrity label that authorizes the downgrading

Does it Help?

- Intuitively appealing for programmers
  - But programmers are still trusted
  - Easy to implement
- Declassification doesn't change the integrity level of a piece of data
  - Noninterference for integrity sublattice still holds
  - Weaker guarantee than needed?
- Could further refine auth(L',L)
  - Restrict declassification to data with particular integrity labels

Dynamic Policies

- Dynamic Principals
  - Identity of principals may change at run time
  - Policy may depend on identity
  - Requires authentication
  - Add a new Java primitive type principal
- Dynamic Labels
  - Policies for dynamic principals
  - May need to examine label dynamically
  - Add a new Java primitive type label

Interface to Outside World

- Should reflect OS file permissions into security types
  - Requires dynamic test of access control
- Legacy code is a problem
  - Interfaces need to be annotated with

Parameterized Classes

- Jif allows classes to be parameterized by labels and principals
  - Code reuse
  - e.g. Containers parameterized by labels

-
  ```
  class MyClass[label L] {
     int{L} x;
  }
  ```

Unix cat in Jif

```
public static void main{}(String{}[]{} args) {
  String filename = args[0];
  final principal p = Runtime.user();
  final label lb;
  lb = new label{p:};
  Runtime[p] runtime = Runtime.getRuntime(p);
  FileInputStream{*lb} fis =
      runtime.openFileRead(filename, lb);
  InputStreamReader{*lb} reader =
      new InputStreamReader{*lb}(fis);
  BufferedReader{*lb} br = new BufferedReader{*lb}(reader);
  PrintStream{*lb} out = runtime.out();
  String line = br.readLine();
  while (line != null) {
    out.println(line);
    line = br.readLine();
  }
}
```

Challenges

- Integrating information flow with other kinds of security
    - Access control
    - Encryption
- Concurrency and distributed prog.
    - Threads can “observer” each other’s behavior
    - Information can leak through scheduler and through synchronization mechanisms.
    - Application of bisimulation & observational equivalence
    - Application of information-flow technology to distributed systems

More Challenges

- Dynamic Security Policies
  - First class principals – dependent types??
  - First class labels
  - Inspect policies dynamically (typecase??)
  - Prove noninterference
- Low-level information-flow anaylses
  - Type preserving compilation
  - Byte-code or assembly level
  - Fine grained analysis

Summary

- Information-flow security is a promising application domain for language technology.
- There are a lot of good results:
  - Basic theory
  - Polymorphism & Inference
  - State & Effects
  - Implementations
- but more are needed!
- There is an excellent survey paper by Sabelfeld and Myers:
  - Language-based Information-flow Security
  - JSAC 21(1) 2003
  - 147 references to other work!

Thanks!

www.cs.cornell.edu/jif