Software Foundations of Security and Privacy (15-316, spring 2017)

Lecture 12: Information Flow (2)

Jean Yang

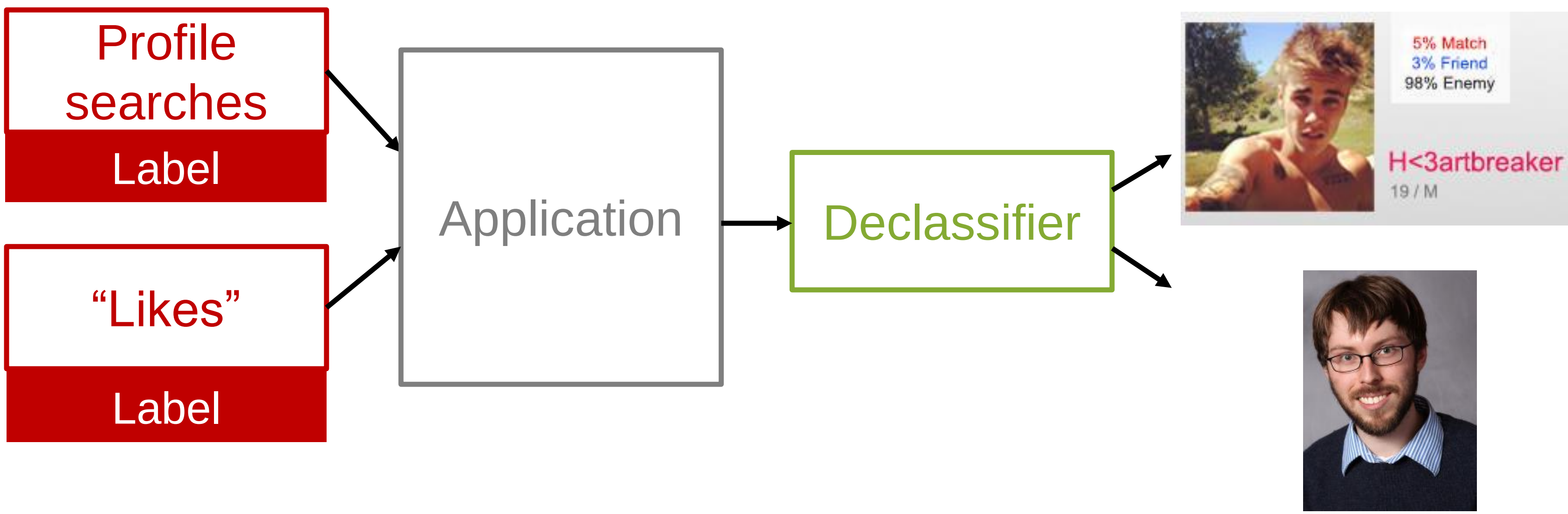
jyang2@andrew.cmu.edu

Last Class

- Motivation for why we need more than access control.
- Process-based decentralized information flow control.
- How we can make reference monitors for information flow, even though it's not a safety property.

Recall DIFC

Model for controlling information flow in systems with *mutual distrust* and *decentralized authority*. Sensitive data is *labelled* and can be *declassified* in a decentralized way.



Problems with Dynamic DIFC

- Non-trivial runtime overheads.
- Required to be conservative, because can only make reference monitors for safety properties.
- Conservative requires us to have all these trusted declassifications all over the place.

What We Want

- ☑ Fine-grained information flow analysis that gives us non-interference.
- ☑ As little run-time overhead as possible.
- ☑ A way to get some static guarantees before we run our programs.

Information flow types give us all of this!

Part One: High-Level Introduction to Language-Level Information Flow Control

Information Flow in Java with Jif

[Myers]

- Jif augments Java types with labels that are *statically* checked*.
  - int {Alice:Bob} x;
  - Object {L} o;
- Subtyping with the $\sqsubseteq$ lattice order determines how differently-labeled values should be combined.
- Type inference allows programmers to omit types.

* Over the years, there has been work to insert additional dynamic checks.

Hello Labels, My Old Friend

Slide from Vitaly Schmatikov.

- Confidentiality constraints: who may read it?
  - {Alice: Bob, Eve} label means that Alice owns this data, and Bob and Eve are permitted to read it
  - {Alice: Charles; Bob: Charles} label means that Alice and Bob own this data but only Charles can read it
- Integrity constraints: who may write it?
  - {Alice ? Bob} label means that Alice owns this data, and Bob is permitted to change it

Labels and Flow

```
int {Alice:Bob} x;
int {Alice:Bob, Charles} y;
x = y;   // Okay, because policy on x is stronger
y = x;   // Bad, because policy on y is weaker
```

- Each owner can specify an independent policy.
- Code running with owner authority can *declassify* data by adding more permissions.
- When a value is read from a slot, it acquires the slot's label.

What About Combining Values?

```
int {Alice:Bob} x;
int {Alice:Bob, Charles} y;
int {??} z;
z = x + y;
```

Q: What label does z need in order for this flow to be allowed?

A: What label does z need in order for this flow to be allowed?

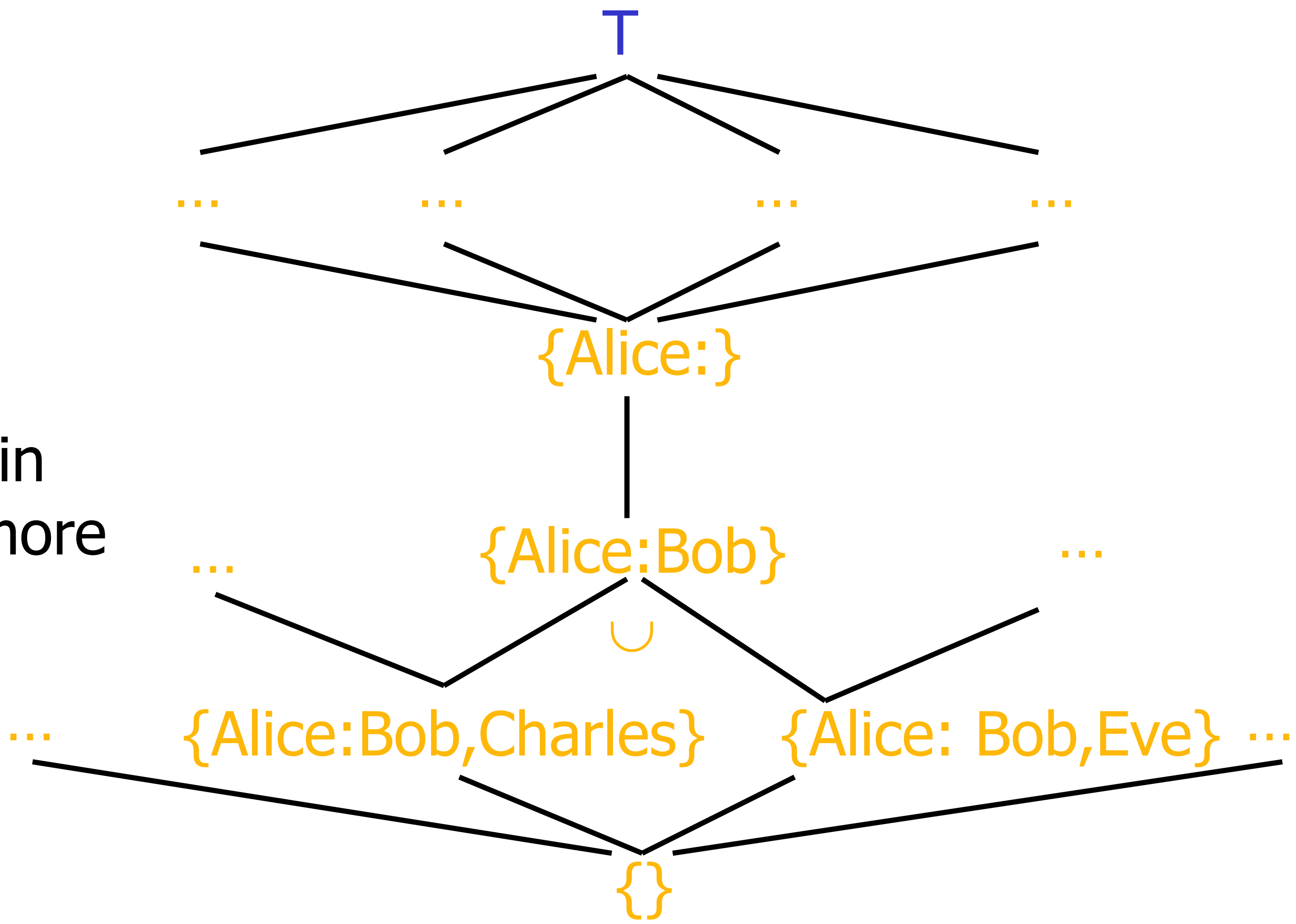
Label Lattice

Slide from Vitaly Schmatikov.

$\sqsubseteq$ order

$\sqcup$ join

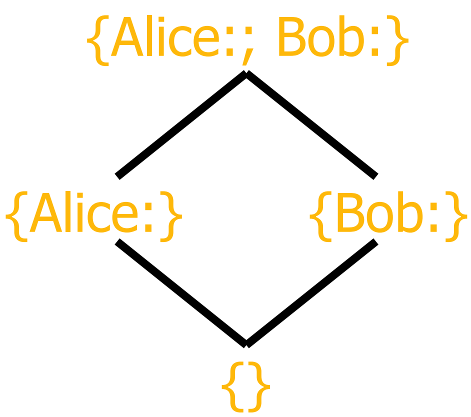
Labels higher in the lattice are more restrictive



Challenge: Implicit Flows

[Zdancewic]

Slide from Vitaly Schmatikov.



PC label

```
int{Alice:} a;
int{Bob:} b;
...
if (a > 0) then {
    b = 4;
}
```

{}

{}∪{Alice:}={Alice:}

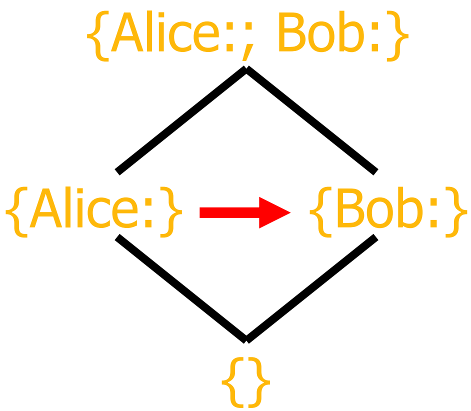
{}

This assignment leaks information contained in program counter (PC)

Challenge: Implicit Flows

[Zdancewic]

Slide from Vitaly Schmatikov.



PC label

```
int{Alice:} a;
int{Bob:} b;
...
if (a > 0) then {
  b = 4;
}
```

{} →

{}∪{Alice:}={Alice:} →

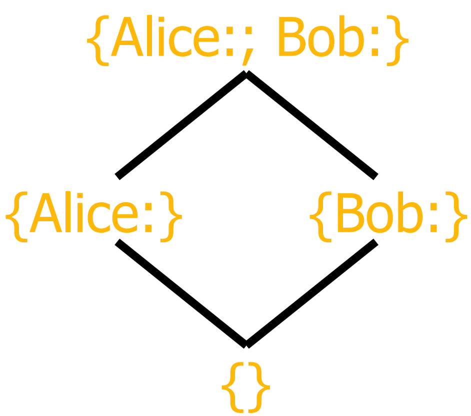
{} →

To assign to variable with label X, must have PC ⊆ X

Challenge: Implicit Flows

[Zdancewic]

Slide from Vitaly Schmatikov.



PC label

```
int{Alice:} a;
int{Bob:} b;
...
if (a > 0) then {
  f(4);
}
```

{} → (before `if`)

{}∪{Alice:}={Alice:} → f(4);

{} → (after `}`)

Effects inside function can leak information about program counter

Part Two: Formalizing the Security Lattice

Security Lattice

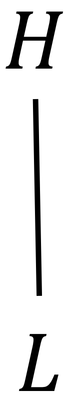
Slide from Matt Fredrikson.

A *security lattice* is a five-tuple $(SC, \leq, \sqcup, \sqcap, \bot)$ where:

- SC is a set of security classes
- $\leq$ is a *partial order* on SC
- $s_1 \sqcup s_2$ is the *least upper bound of* s_1 and s_2, $s_{1,2} \leq s_1 \sqcup s_2$, and $\forall s \in SC. s_{1,2} \leq s \Rightarrow s_1 \sqcup s_2 \leq s$
- $s_1 \sqcap s_2$ is the *greatest lower bound of* s_1 and s_2, $s_1 \sqcap s_2 \leq s_{1,2}$, and $\forall s \in SC. s_{1,2} \leq s \Rightarrow s \leq s_1 \sqcap s_2$
- $\bot$ is the least element of SC

A Simple Lattice for Secrecy

Slide from Matt Fredrikson.



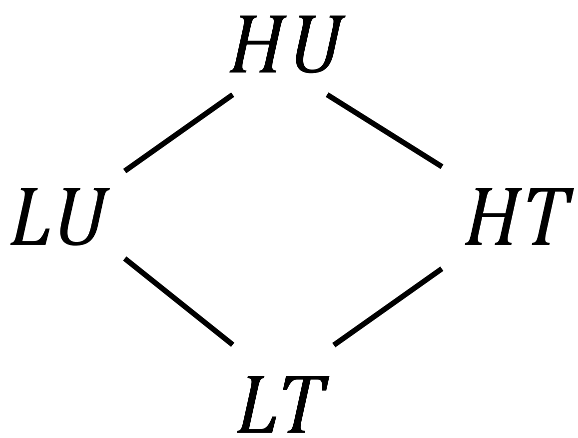
Policy: no high-security flows to low-variables.

- H is "high" and L is "low"
- $L \leq H, \neg(H \leq L)$, and L is $\perp$

The partial order $\leq$ means "can flow to."

Secrecy and Integrity

Slide from Matt Fredrikson.



Policy: no high flows to low, no untrusted flows to trusted

- H is "high," L is "low," U is "untrusted," and T is "trusted"
- $T \leq U, \neg(U \leq T)$

Part Three: A Type System for Information Flow

A Simple Imperative Language

Slide from Matt Fredrikson.

Arithmetic expressions

$$a \in AExp ::= n \in \mathbb{Z} \mid x \in \mathbf{Var}$$

$$\mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2$$

Boolean expressions

$$b \in BExp ::= \mathbf{T} \mid \mathbf{F} \mid \neg b \mid b_1 \wedge b_2 \mid a_1 = a_2 \mid a_1 \leq a_2$$

Commands

$$c \in Com \quad ::= \mathbf{skip} \mid x := a \mid c_1 ; c_2$$

$$\mid \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2$$

$$\mid \mathbf{while}\ b\ \mathbf{do}\ c$$

Expression Evaluation

Slide from Matt Fredrikson.

States are mappings $\sigma: \mathbf{Var} \mapsto \mathbb{Z}$

Expression evaluation happens with the *big-step relation* $\langle \sigma, a \rangle \Downarrow n$

$$\frac{}{\langle \sigma, n \rangle \Downarrow n} \qquad \frac{}{\langle \sigma, x \rangle \Downarrow \sigma(x)}$$

$$\frac{\langle \sigma, a_1 \rangle \Downarrow n_1 \qquad \langle \sigma, a_2 \rangle \Downarrow n_2 \qquad n = n_1 \mathbf{op}\, n_2}{\langle \sigma, a_1 \mathbf{op}\, a_2 \rangle \Downarrow n}$$

Command Evaluation

Slide from Matt Fredrikson.

Big-step relation $\langle \sigma_1, c \rangle \Downarrow \sigma_2$

$$\frac{\langle \sigma, a \rangle \Downarrow n}{\langle \sigma, x := a \rangle \Downarrow \sigma[x \mapsto n]}$$

$$\frac{\langle \sigma_1, c \rangle \Downarrow \sigma_2}{\langle \sigma, \mathbf{skip}; c \rangle \Downarrow \sigma_2}$$

$$\frac{\langle \sigma_1, c_1 \rangle \Downarrow \sigma_1' \quad \langle \sigma_1', c_2 \rangle \Downarrow \sigma_2}{\langle \sigma_1, c_1; c_2 \rangle \Downarrow \sigma_2}$$

$$\frac{\langle \sigma, b \rangle \Downarrow \mathbf{T} \quad \langle \sigma, c_1 \rangle \Downarrow \sigma_2}{\langle \sigma, \mathbf{if}\ b\ \mathbf{then}\ c_1\, \mathbf{else}\ c_2 \rangle \Downarrow \sigma_2}$$

$$\frac{\langle \sigma, b \rangle \Downarrow \mathbf{F} \quad \langle \sigma, c_2 \rangle \Downarrow \sigma_2}{\langle \sigma, \mathbf{if}\ b\ \mathbf{then}\ c_1\, \mathbf{else}\ c_2 \rangle \Downarrow \sigma_2}$$

$$\frac{\langle \sigma, b \rangle \Downarrow \mathbf{F}}{\langle \sigma, \mathbf{while}\ b\ \mathbf{do}\ c \rangle \Downarrow \sigma}$$

$$\frac{\langle \sigma_1, b \rangle \Downarrow \mathbf{T} \quad \langle \sigma_1, c \rangle \Downarrow \sigma_1' \quad \langle \sigma_1', while\ b\ do\ c \rangle \Downarrow \sigma_2}{\langle \sigma_1, \mathbf{while}\ b\ \mathbf{do}\ c \rangle \Downarrow \sigma_2}$$

Type Environment Γ

Slide from Matt Fredrikson.

Let $L = (SC, \leq, \sqcup, \sqcap, \bot)$ be a security lattice. A *type environment* $\Gamma\colon \mathbf{Var} \mapsto SC$ for a program c maps each variable in c to a label.

Additionally, Γ contains an additional mapping for the program counter label $\mathbf{pc}$.

- $\Gamma \vdash e : \ell$ means expression e has label ℓ under Γ
- $\Gamma \vdash c$ means c is well-typed under Γ
- Environment $(\Gamma, x :: \ell)$ gives x type ℓ, preserves rest of Γ

Goal: Noninterference

Slide from Matt Fredrikson.

State Equivalence

Abbreviated as

$\sigma_1 \approx_\ell \sigma_2$

Two states σ_1, σ_2 are ℓ-equivalent to an observer of class $\ell \in SC$ under Γ, written $\boxed{\sigma_1 \approx_{\ell,\Gamma} \sigma_2}$ if and only if

$$\forall x \in \mathbf{Var}.\, \Gamma(x) \leq \ell \Rightarrow \sigma_1(x) = \sigma_2(x)$$

Noninterference

A program c satisfies noninterference at class ℓ under Γ if ℓ-equivalent initial states lead to ℓ-equivalent final states:

$$\forall \sigma_1, \sigma_2.\, \boxed{\sigma_{2\approx_\ell}\sigma_2} \wedge \langle \sigma_1, c \rangle \Downarrow \sigma_1' \wedge \langle \sigma_2, c \rangle \Downarrow \sigma_2' \Rightarrow \boxed{\sigma_1' \approx_\ell \sigma_2'}$$

Initial states are state equivalent

Final states are state equivalent

Typing Rules: Expressions

Slide from Matt Fredrikson.

Var $\dfrac{}{\Gamma \vdash x : \Gamma(x)}$ Int $\dfrac{}{\Gamma \vdash n : \bot}$ True $\dfrac{}{\Gamma \vdash \mathbf{T} : \bot}$ False $\dfrac{}{\Gamma \vdash \mathbf{F} : \bot}$

$$\text{Bin}\ \dfrac{\Gamma \vdash a_1 : \ell_1 \qquad \Gamma \vdash a_2 : \ell_2}{\Gamma \vdash a_1\, op\ a_2 : \ell_1 \sqcup \ell_2}$$

Example

$$\boxed{5 \leq 6 + x, \Gamma = \mathrm{x} :: \mathrm{H}}$$

$$\dfrac{\dfrac{}{\Gamma \vdash 5 : L}\text{Int} \qquad \dfrac{\dfrac{}{\Gamma \vdash 6 : L}\text{Int} \qquad \dfrac{}{\Gamma \vdash x : H}\text{Var}}{\Gamma \vdash 6 + x :\ H}\text{Bin}}{\Gamma \vdash 5 \leq 6 + x :\ H}\text{Bin}$$

Typing Rules: Commands

Slide from Matt Fredrikson.

Skip $\dfrac{}{\Gamma \vdash \mathbf{skip}}$

Asgn $\dfrac{\Gamma \vdash a : \ell \quad \ell \sqcup \Gamma(\mathbf{pc}) \leq \Gamma(x)}{\Gamma \vdash x := a}$

Comp $\dfrac{\Gamma \vdash c_1 \quad \Gamma \vdash c_2}{\Gamma \vdash c_1 ; c_2}$

While $\dfrac{\Gamma \vdash b : \ell \quad \ell' = \Gamma(\mathbf{pc}) \sqcup \ell \quad \Gamma, \mathbf{pc} :: \ell' \vdash c}{\Gamma \vdash \mathbf{while}\ b\ \mathbf{do}\ c}$

If $\dfrac{\Gamma \vdash b : \ell \quad \ell' = \Gamma(pc) \sqcup \ell \quad \Gamma, \mathbf{pc} :: \ell' \vdash c_1 \quad \Gamma, \mathbf{pc} :: \ell' \vdash c\_2}{\Gamma \vdash \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2}$

Command Typing Example

Slide from Matt Fredrikson.

$$\boxed{\Gamma = p :: H, g :: L, o :: L, pc :: L}$$

$$\text{If}\ \dfrac{\text{Bin}\ \dfrac{\ldots}{\Gamma \vdash p = g : H} \quad H = \Gamma(pc) \sqcup H \quad \text{Asgn}\ \dfrac{\text{Int}\ \dfrac{}{\Gamma \vdash 1 : L} \quad L \sqcup H \leq \Gamma(o)}{\Gamma, \mathbf{pc} :: H \vdash o := 1}}{\Gamma \vdash \mathbf{if}\ p = g\ \mathbf{then}\ o := 1\ \mathbf{else}\ o := 2}$$

Command Typing Rules

$$\text{Skip}\ \dfrac{}{\Gamma \vdash \mathbf{skip}}$$

$$\text{Asgn}\ \dfrac{\Gamma \vdash a : \ell \quad \ell \sqcup \Gamma(\mathbf{pc}) \leq \Gamma(x)}{\Gamma \vdash x := a}$$

$$\text{Comp}\ \dfrac{\Gamma \vdash c_1 \quad \Gamma \vdash c_2}{\Gamma \vdash c_1 ; c_2}$$

$$\text{While}\ \dfrac{\Gamma \vdash b : \ell \quad \ell' = \Gamma(\mathbf{pc}) \sqcup \ell \quad \Gamma, \mathbf{pc} :: \ell' \vdash c}{\Gamma \vdash \mathbf{while}\ b\ \mathbf{do}\ c}$$

$$\text{If}\ \dfrac{\Gamma \vdash b : \ell \quad \ell' = \Gamma(pc) \sqcup \ell \quad \Gamma, \mathbf{pc} :: \ell' \vdash c_1 \quad \Gamma, \mathbf{pc} :: \ell' \vdash c\_2}{\Gamma \vdash \mathbf{if}\ b\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2}$$

Command Typing Example

Slide from Matt Fredrikson.

$$\boxed{\Gamma = p :: H, g :: L, o :: L, pc :: L}$$

$$\text{If}\ \dfrac{\text{Bin}\ \dfrac{\ldots}{\Gamma \vdash p = g : H} \quad H = \Gamma(pc) \sqcup H \quad \text{Asgn}\ \dfrac{\text{Int}\ \dfrac{}{\Gamma \vdash 1 : L} \quad L \sqcup H \leq \Gamma(o)}{\Gamma, \mathbf{pc} :: H \vdash o := 1}}{\Gamma \vdash \mathbf{if}\ p = g\ \mathbf{then}\ o := 1\ \mathbf{else}\ o := 2}$$

- Doesn't work.
- Guard raises the **pc** label and Asgn propagates it.
- What about $\mathbf{if}\ p = g\ \mathbf{then}\ o := 1\ \mathbf{else}\ o := 1$?

Part Three: Proving Soundness

Soundness

[Volpano, Smith, Irvine '96]

Slide from Matt Fredrikson.

The type system is sound if whenever conditions 1-3 hold for program c and type environment Γ, then c has noninterference (i.e., the final states $\sigma_1' \approx_\ell \sigma_2'$ for any starting states $\sigma_1 \approx_\ell \sigma_2$).

1. $\Gamma \vdash c$
2. $\langle \sigma_1, c \rangle \Downarrow \sigma_1', \langle \sigma_2, c \rangle \Downarrow \sigma_2'$
3. $\sigma_1 \approx_\ell \sigma_2$

Two Key Lemmas

Slide from Matt Fredrikson.

Lemma (Simple Security). Expressions never read variables above their typed class: if $\Gamma \vdash e : \ell$, then for every variable x appearing in e, $\Gamma(x) \leq \ell$.

Lemma (Confinement). Commands never write to variables below $\mathbf{pc}$'s typed class: if $\Gamma \vdash c$, then for every variable x assigned in c, $\Gamma(\mathbf{pc}) \leq \Gamma(x)$.

Proof: Simple Security

Slide from Matt Fredrikson.

Lemma (Simple Security). If $\Gamma \vdash e\!: \ell$, then for every variable x appearing in e, $\Gamma(x) \le \ell$.

Proof by induction on the structure of e :

- Base cases $n, \mathbf{T},$ and $\mathbf{F}$ are trivial.
- Base case x: we have $\Gamma \vdash x\!: \ell$.

  By Var, $\Gamma(x) = \ell$, so $\Gamma(x) \le \ell$.
- Case $e_1\, \mathbf{op}\, e_2$: by Bin, we have $\Gamma \vdash e_1\!: \ell_1$ and $\Gamma \vdash e_2\!: \ell_2$. By induction, we have $\forall x \in e_1.\ \Gamma(x) \le \ell_1$ and $\forall x \in e_2.\ \Gamma(x) \le \ell_2$. Then $\Gamma(x) \le \ell_1 \sqcup \ell_2 = \ell$ for all $e = e_1\, \mathbf{op}\, e_2$. $\square$

Proof: Confinement

Slide from Matt Fredrikson.

Lemma (Confinement). if $\Gamma \vdash c$, then for every variable x assigned in c, $\Gamma(\mathbf{pc}) \leq \Gamma(x)$.

Proof by induction on the structure of c :

- Base case $\mathbf{skip}$ is trivial.
- Base case $x \coloneqq a$: we have $\Gamma \vdash a : \ell$.
  By Asgn, $\ell \sqcup \Gamma(\mathbf{pc}) \leq \Gamma(x)$, so $\Gamma(\mathbf{pc}) \leq \Gamma(x)$.
- Case $c_1; c_2$ follows directly by induction.
- Case $\mathbf{while}\ b\ \mathbf{do}\ c$: suppose $\Gamma \vdash b : \ell$.
  By While, we have that $\Gamma, \mathbf{pc} :: (\ell \sqcup \mathbf{\Gamma}(\mathbf{pc})) \vdash c$.
  By induction, we have that $\forall x \in c.\, \ell \sqcup \Gamma(\mathbf{pc}) \leq \Gamma(x)$.
  By $\leq$-transitivity, $\forall x \in c.\, \Gamma(\mathbf{pc}) \leq \Gamma(x)$.
- The case for $\mathbf{if}$ is similar to $\mathbf{while}$. $\square$

Proof Sketch: Soundness

Slide from Matt Fredrikson.

Theorem (Soundness). The type system is sound if whenever conditions 1-3 hold for program c and type environment Γ, then c has noninterference (i.e., the final states $\sigma_1' \approx_\ell \sigma_2'$ for any starting states $\sigma_1 \approx_\ell \sigma_2$).

1. $\Gamma \vdash c$
2. $\langle \sigma_1, c \rangle \Downarrow \sigma_1', \langle \sigma_2, c \rangle \Downarrow \sigma_2'$
3. $\sigma_1 \approx_\ell \sigma_2$

Proof by induction on the derivation of $\langle \sigma_1, c \rangle \Downarrow \sigma_1'$:

- Use Simple Security to argue about identical evaluation.
- Use Confinement to argue about ℓ-equivalent updates.

Example: **while**

Theorem (Soundness). Want following conditions:

1. $\Gamma \vdash c$
2. $\langle \sigma_1, c\rangle \Downarrow \sigma_1', \langle \sigma_2, c\rangle \Downarrow \sigma_2'$
3. $\sigma_1 \approx_\ell \sigma_2$

Suppose $\langle \sigma_1, \mathbf{while}\ b\ \mathbf{do}\ c\rangle \Downarrow \sigma_1'$ and typing ends with:

$$\frac{\Gamma \vdash b : \ell_1 \quad \ell_2 = \Gamma(\mathbf{pc}) \sqcup \ell_1 \quad \Gamma, \mathbf{pc} :: \ell_2 \vdash c}{\ldots \mathbf{ile}\ b\ \mathbf{do}\ c}$$

$$\frac{\langle \sigma_{1,2}, b\rangle \Downarrow \mathbf{T} \qquad \langle \sigma_{1,2}, c\rangle \Downarrow \sigma_{1,2}'' \qquad \langle \sigma_{1,2}'', \mathbf{while}\ b\ \mathbf{do}\ c\rangle \Downarrow \sigma_{1,2}'}{\langle \sigma_{1,2}, \mathbf{while}\ b\ \mathbf{do}\ c\rangle \Downarrow \sigma_{1,2}'}$$

Case $l_2 \le$ … ow memory):

- By Sim … for all x in b.
- By (3), … , so $\langle \sigma_1, b\rangle \Downarrow v$ and $\langle \sigma_2, b\rangle \Downarrow v$
- If $v = \mathbf{F}$, th… $\sigma_1 = \sigma_1'$ and $\sigma_2 = \sigma_2'$. Invoke (3).
- If $v = \mathbf{T}$, then $\sigma_1'' \approx_\ell \sigma_2''$ by induction. Then $\sigma_1' \approx_\ell \sigma_2'$ also by induction.

Example: **while**

Theorem (Soundness). Want following conditions:

1. $\Gamma \vdash c$
2. $\langle \sigma_1, c\rangle \Downarrow \sigma_1', \langle \sigma_2, c\rangle \Downarrow \sigma_2'$
3. $\sigma_1 \approx_\ell \sigma_2$

Suppose $\langle \sigma_1, \textbf{while}\ b\ \textbf{do}\ c\rangle \Downarrow \sigma_1'$ and typing ends with:

$$\text{While}\ \frac{\Gamma \vdash b : \ell_1 \quad \ell_2 = \Gamma(\textbf{pc}) \sqcup \ell_1 \quad \Gamma, \textbf{pc} :: \ell_2 \vdash c}{\Gamma \vdash \textbf{while}\ b\ \textbf{do}\ c}$$

Case $l_2 > \ell$ (condition cannot flow into low memory):

- By Confinement, $\ell_1 \leq \Gamma(x)$ for all x assigned in c.
- For x assigned in c, $\neg(\Gamma(x) \leq \ell)$.
- For every x in c where $\Gamma(x) \leq \ell$, $\sigma_{1,2}(x) = \sigma_{1,2}'(x)$.
- By (3), we have $\sigma_1 \approx_\ell \sigma_2'$. $\square$

Discussion Questions

- What kinds of guarantees can language-based information flow provide?
- What are the tradeoffs of static information flow analysis?
- This work came *before* the Flume work. Why did people become interested in coarser-grained information flow?