

18732: Secure Software Systems

# Language-based Security: Information Flow Control

Anupam Datta

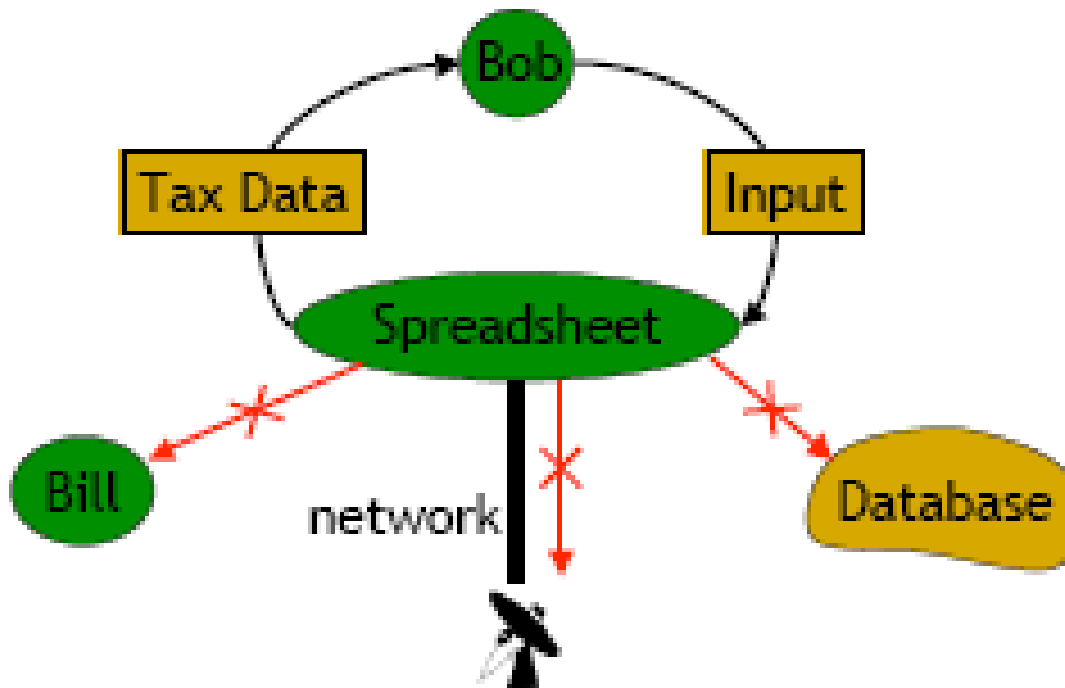
CMU  
Fall 2010

# Lecture Outline

## Information Flow Control (IFC)

- Security definition
  - Non-interference [Goguen-Meseguer82]
- Language-based enforcement
  - Type system [Volpano-Smith-Irvine96] based on prior work [Denning-Denning77]

# IFC in Tax Preparation Software



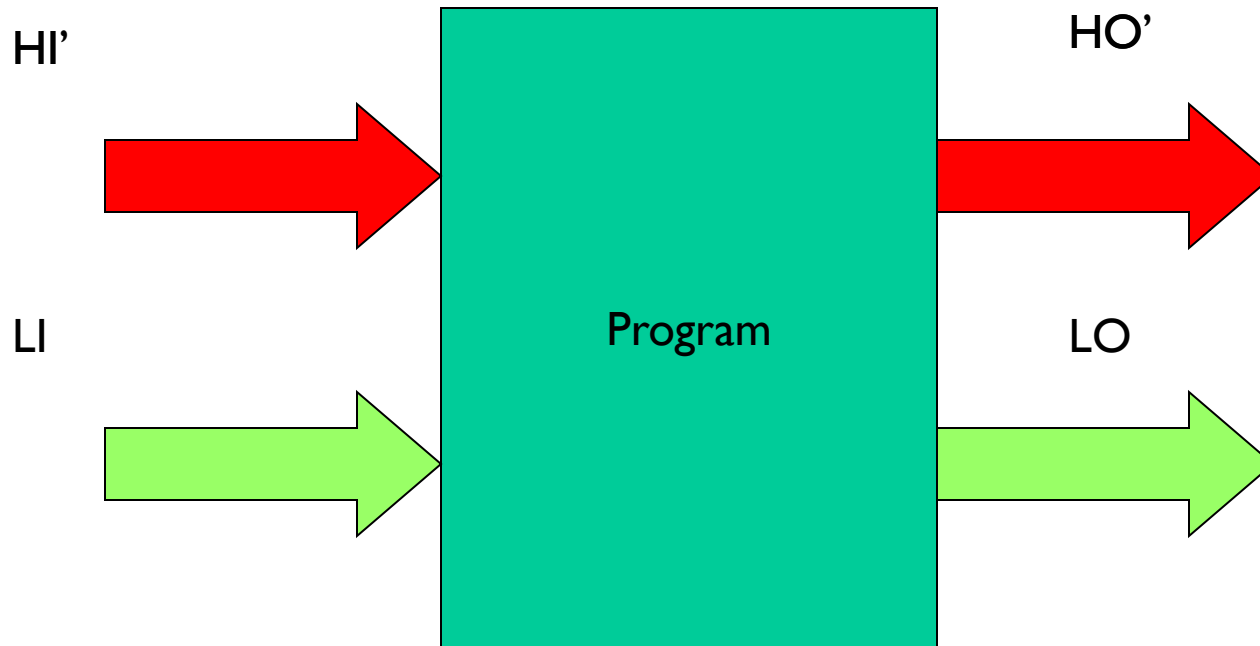
# Definition of Security

- Non-interference (idea)

Security levels:

H: Classified

L: Unclassified



No information flows from high inputs to low outputs

# Example

if  $x = 1$  then  $y := 1$  else  $y := 0$

x	y	NI
H	H	Yes
L	L	Yes
H	L	No
L	H	Yes

# Specification and Enforcement

- Approach
  - Use a typed programming language
  - Types represent *security levels*
    - H, L, ...
  - Sub-typing captures partial order among security levels
    - $L \leq H$
  - Type system captures allowed information flows
  - Soundness theorem
    - Well-typed programs satisfy non-interference

# Language Definition

- Syntax
- Type System
- Operational Semantics
  
- Soundness Theorem
  - Well typed programs satisfy non-interference

# Syntax (I)

We consider a core block-structured language described below. It consists of phrases, which are either expressions  $e$  or commands  $c$ :

$$\begin{array}{ll}
 (\textit{phrases}) & p ::= e \mid c \\
 (\textit{expressions}) & e ::= x \mid l \mid n \mid e + e' \mid e - e' \mid e = e' \mid e < e' \\
 (\textit{commands}) & c ::= e := e' \mid c; c' \mid \mathbf{if } e \mathbf{ then } c \mathbf{ else } c' \mid \\
 & \mathbf{while } e \mathbf{ do } c \mid \mathbf{letvar } x := e \mathbf{ in } c
 \end{array}$$

Metavariable  $x$  ranges over identifiers,  $l$  over *locations* (addresses), and  $n$  over integer literals. Integers are the only values. We use 0 for false and 1 for true, and assume that locations are well ordered.

# Syntax (II)

The types of the core language are stratified as follows.

(*data types*)  $\tau ::= s$

(*phrase types*)  $\rho ::= \tau \mid \tau \textit{ var} \mid \tau \textit{ cmd}$

Metavariable  $s$  ranges over the set  $SC$  of security classes, which is assumed to be partially ordered by  $\leq$ . Type  $\tau \textit{ var}$  is the type of a variable and  $\tau \textit{ cmd}$  is the type of a command.

We will focus on the special case where type  $\tau$  is

either  $H$  or  $L$  and  $L \leq H$

# Type System (I)

- Typing judgment

$$\lambda; \gamma \vdash p : \rho$$

where  $\lambda$  is a *location typing* and  $\gamma$  is an *identifier typing*. The judgment means that phrase  $p$  has type  $\rho$ , assuming  $\lambda$  prescribes types for locations in  $p$  and  $\gamma$  prescribes types for any free identifiers in  $p$ . An identifier typing is a finite function mapping identifiers to  $\rho$  types;  $\gamma(x)$  is the  $\rho$  type assigned to  $x$  by  $\gamma$ . Also,  $\gamma[x : \rho]$  is a modified identifier typing that assigns type  $\rho$  to  $x$  and assigns type  $\gamma(x')$  to any identifier  $x'$  other than  $x$ . A location typing is a finite function mapping locations to  $\tau$  types. The notational conventions for location typings are similar to those for identifier typings.

# Type system (II)

(INT)	$\lambda; \gamma \vdash n : \tau$
(VAR)	$\lambda; \gamma \vdash x : \tau \text{ var}$ if $\gamma(x) = \tau \text{ var}$
(VARLOC)	$\lambda; \gamma \vdash l : \tau \text{ var}$ if $\lambda(l) = \tau$
(ARITH)	$\frac{\lambda; \gamma \vdash e : \tau, \quad \lambda; \gamma \vdash e' : \tau}{\lambda; \gamma \vdash e + e' : \tau}$
(R-VAL)	$\frac{\lambda; \gamma \vdash e : \tau \text{ var}}{\lambda; \gamma \vdash e : \tau}$
(ASSIGN)	$\frac{\lambda; \gamma \vdash e : \tau \text{ var}, \quad \lambda; \gamma \vdash e' : \tau}{\lambda; \gamma \vdash e := e' : \tau \text{ cmd}}$

# Type System (III)

(COMPOSE)	$\frac{\lambda; \gamma \vdash c : \tau \text{ cmd}, \quad \lambda; \gamma \vdash c' : \tau \text{ cmd}}{\lambda; \gamma \vdash c; c' : \tau \text{ cmd}}$
(IF)	$\frac{\lambda; \gamma \vdash e : \tau, \quad \lambda; \gamma \vdash c : \tau \text{ cmd}, \quad \lambda; \gamma \vdash c' : \tau \text{ cmd}}{\lambda; \gamma \vdash \mathbf{if } e \mathbf{ then } c \mathbf{ else } c' : \tau \text{ cmd}}$
(WHILE)	$\frac{\lambda; \gamma \vdash e : \tau, \quad \lambda; \gamma \vdash c : \tau \text{ cmd}}{\lambda; \gamma \vdash \mathbf{while } e \mathbf{ do } c : \tau \text{ cmd}}$
(LETVAR)	$\frac{\lambda; \gamma \vdash e : \tau, \quad \lambda; \gamma[x : \tau \text{ var}] \vdash c : \tau' \text{ cmd}}{\lambda; \gamma \vdash \mathbf{letvar } x := e \mathbf{ in } c : \tau' \text{ cmd}}$

# Example

if  $x = 1$  then  $y := 1$  else  $y := 0$

x	y	NI
H	H	Yes
L	L	Yes
H	L	No
L	H	Yes

Will justify rows 1 & 2

# Example with types

For example, suppose  $\gamma(x) = \gamma(y) = H \text{ var}$ . By the preceding typing rule for assignment, we have  $\gamma \vdash y := 1 : H \text{ cmd}$  and  $\gamma \vdash y := 0 : H \text{ cmd}$ . This means that each statement can be placed in a context where high information is implicitly known through the guard of a conditional statement. An example is **if  $x = 1$  then  $y := 1$  else  $y := 0$** . With  $\tau = H$ , the secure flow typing rule for conditionals gives

$$\gamma \vdash \mathbf{if } x = 1 \mathbf{ then } y := 1 \mathbf{ else } y := 0 : H \text{ cmd}$$

Key rules used are (ASSIGN) and (IF)

# Type System (IV)

(BASE)	$\frac{\tau \leq \tau'}{\vdash \tau \subseteq \tau'}$
(REFLEX)	$\vdash \rho \subseteq \rho$
(TRANS)	$\frac{\vdash \rho \subseteq \rho', \vdash \rho' \subseteq \rho''}{\vdash \rho \subseteq \rho''}$
(CMD <sup>-</sup> )	$\frac{\vdash \tau \subseteq \tau'}{\vdash \tau' \text{ cmd} \subseteq \tau \text{ cmd}}$
(SUBTYPE)	$\frac{\lambda; \gamma \vdash p : \rho, \vdash \rho \subseteq \rho'}{\lambda; \gamma \vdash p : \rho'}$

Figure 3. Subtyping rules

# Example

if  $x = 1$  then  $y:=1$  else  $y:=0$

x	y	NI
H	H	Yes
L	L	Yes
H	L	No
L	H	Yes

Will justify rows 3 & 4

# Example with types

- Suppose  $x: L$  var and  $y: H$  var  $L \leq H$ 
  1. Use (ASSIGN), (CMD-), (SUBTYPE) to infer  $(y:=1): L$  cmd and  $(y:=0): L$  cmd
  2. Now use (IF) rule
- $x: H$  var and  $y: L$  var is not well-typed as expected

# Operational Semantics (I)

- $\mu$  is memory: a function from locations to values
- $\mu(l)$  is contents of location  $l$
- Judgments
  1. Evaluating expression  $e$  in memory  $\mu$  yields value  $n$   
$$\mu \vdash e \Rightarrow n$$
  2. Evaluating command  $c$  in memory  $\mu$  yields memory  $\mu'$   
$$\mu \vdash c \Rightarrow \mu'$$

Program executes by evaluating expressions and commands

# Operational Semantics (II)

$$\text{(BASE)} \quad \mu \vdash n \Rightarrow n$$

$$\text{(CONTENTS)} \quad \mu \vdash l \Rightarrow \mu(l) \quad \text{if } l \in \text{dom}(\mu)$$

$$\text{(ADD)} \quad \frac{\mu \vdash e \Rightarrow n, \quad \mu \vdash e' \Rightarrow n'}{\mu \vdash e + e' \Rightarrow n + n'}$$

$$\text{(UPDATE)} \quad \frac{\mu \vdash e \Rightarrow n, \quad l \in \text{dom}(\mu)}{\mu \vdash l := e \Rightarrow \mu[l := n]}$$

$$\text{(SEQUENCE)} \quad \frac{\mu \vdash c \Rightarrow \mu', \quad \mu' \vdash c' \Rightarrow \mu''}{\mu \vdash c; c' \Rightarrow \mu''}$$

$$\text{(BRANCH)} \quad \frac{\mu \vdash e \Rightarrow 1, \quad \mu \vdash c \Rightarrow \mu'}{\mu \vdash \text{if } e \text{ then } c \text{ else } c' \Rightarrow \mu'}$$

$$\mu \vdash e \Rightarrow 0, \quad \mu \vdash c' \Rightarrow \mu'$$

$$\mu \vdash \text{if } e \text{ then } c \text{ else } c' \Rightarrow \mu'$$

# Operational Semantics (III)

$$\text{(LOOP)} \quad \frac{\mu \vdash e \Rightarrow 0}{\mu \vdash \mathbf{while} \ e \ \mathbf{do} \ c \Rightarrow \mu}$$

$$\frac{\begin{array}{l} \mu \vdash e \Rightarrow 1, \\ \mu \vdash c \Rightarrow \mu', \\ \mu' \vdash \mathbf{while} \ e \ \mathbf{do} \ c \Rightarrow \mu'' \end{array}}{\mu \vdash \mathbf{while} \ e \ \mathbf{do} \ c \Rightarrow \mu''}$$

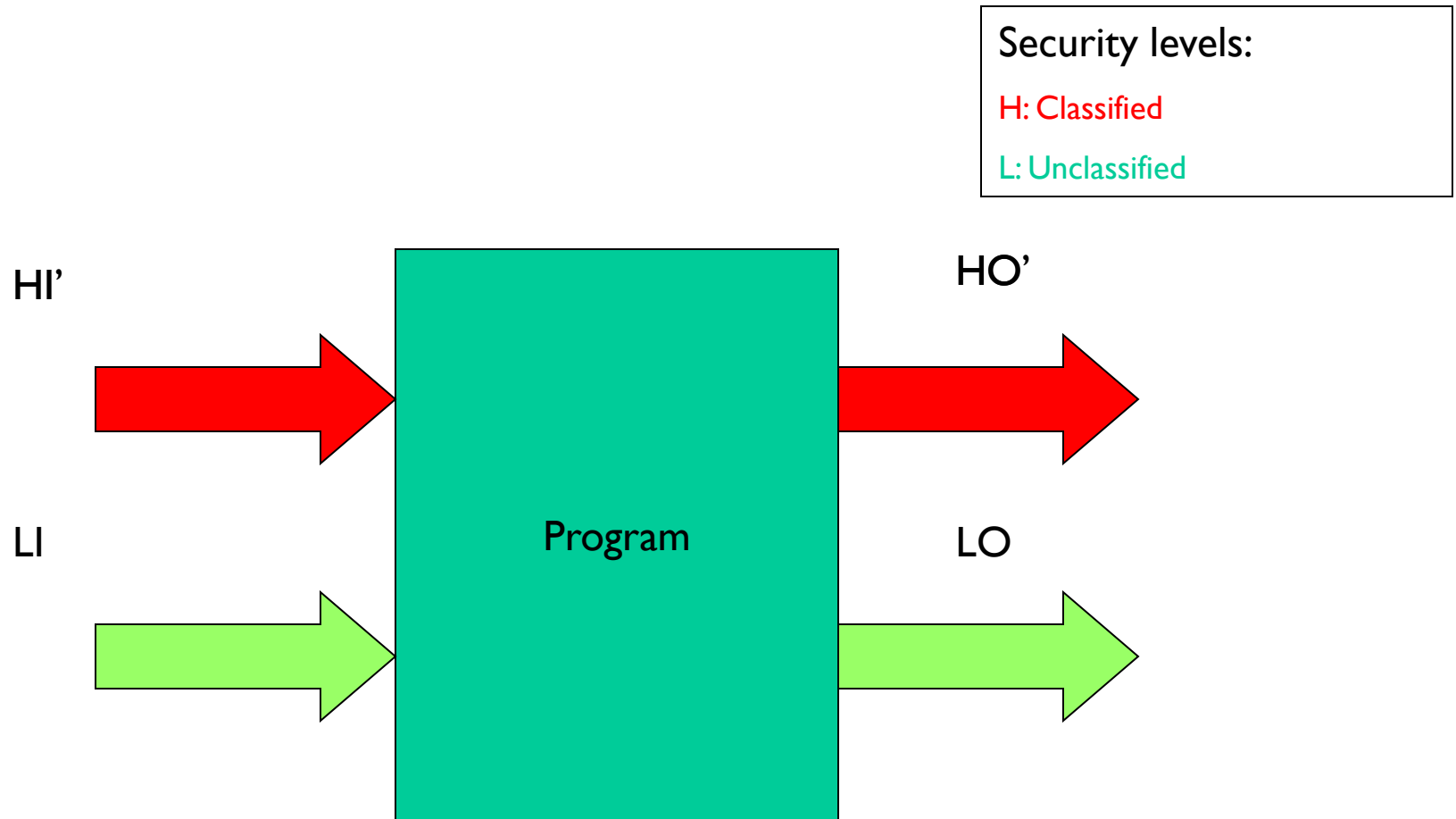
$$\text{(BINDVAR)} \quad \frac{\begin{array}{l} \mu \vdash e \Rightarrow n, \\ l \text{ is the first location not in } \mathit{dom}(\mu), \\ \mu[l := n] \vdash [l/x]c \Rightarrow \mu' \end{array}}{\mu \vdash \mathbf{letvar} \ x := e \ \mathbf{in} \ c \Rightarrow \mu' - l}$$

# Soundness Theorem

Theorem 6.8 (*Type Soundness*) Suppose

- (a)  $\lambda \vdash c : \rho$  *c is well-typed*
  - (b)  $\mu \vdash c \Rightarrow \mu'$  *execution one*
  - (c)  $v \vdash c \Rightarrow v'$  *execution two*
  - (d)  $\text{dom}(\mu) = \text{dom}(v) = \text{dom}(\lambda)$
  - (e)  $v(l) = \mu(l)$  for all  $l$  such that  $\lambda(l) \leq \tau$  *the same low inputs*
- Then  $v'(l) = \mu'(l)$  for all  $l$  such that  $\lambda(l) \leq \tau$ . *the same low outputs*

# Recall Non-interference

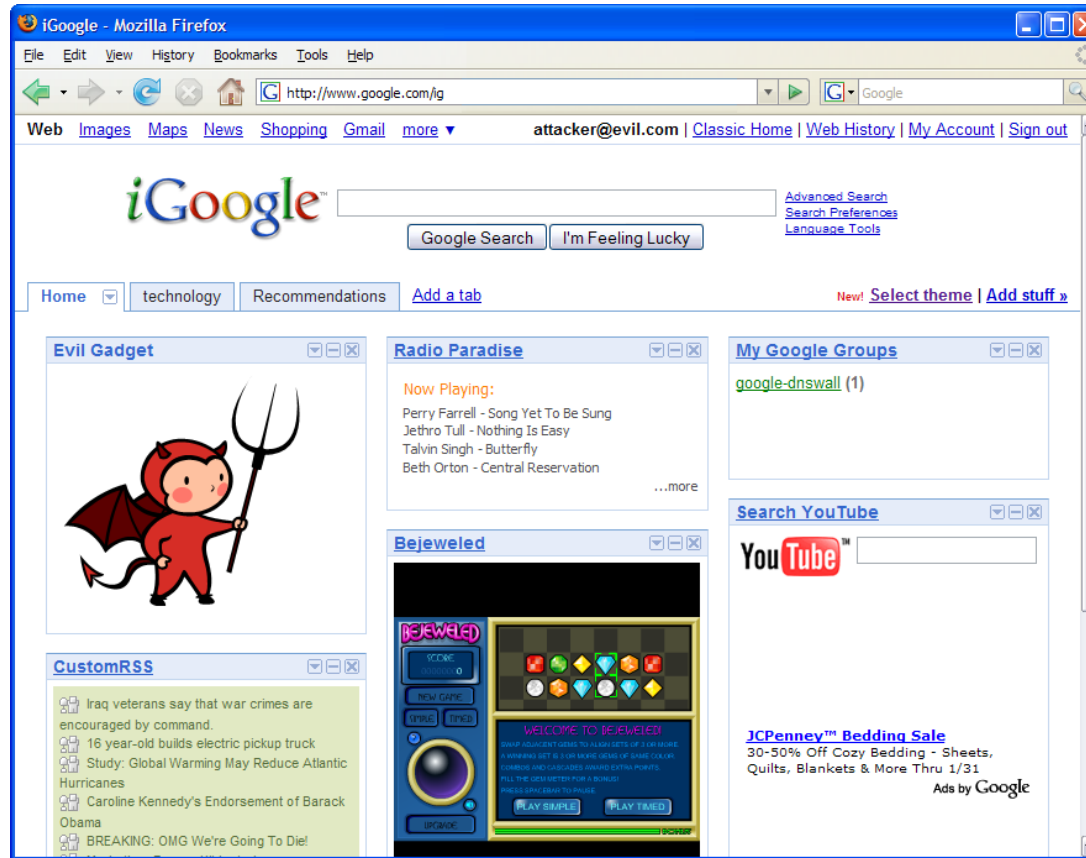


No information flows from high inputs to low outputs

# Practical Languages for IFC

- **Jif** [Liskov-Myers *et al.*]
  - Java + information flow
  - <http://www.cs.cornell.edu/jif/>
- Flow Caml [Pottier-Simonet]
  - Extends OCaml language with type system for tracing information flow
  - <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.11.2104>

# Web Security: A Domain for IFC



Brendan Eich, Chief Technology Officer, Mozilla Corp.

Improving JavaScript's Default Security Model with Information Flow, CSF

2009 Invited talk