Language Based Information Flow Security

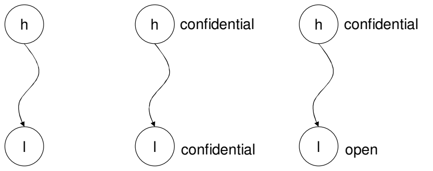
Andrei Sabelfield, Andrew C. Myers

Presentation: Ashish Kundu
ashishk@cs.purdue.edu

Outline

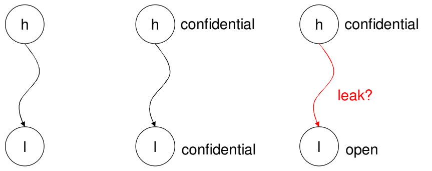
- Security requirements
- Information flow – background
- Language-based information flow
- Open challenges
- Discussion
- Conclusion

Information flow?



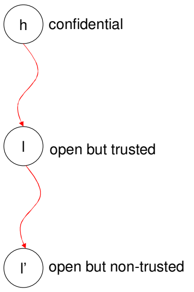
data flow

Information flow?



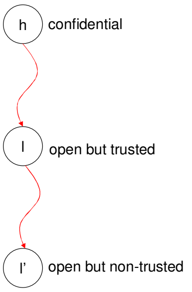
data flow

Information flow?



data flow

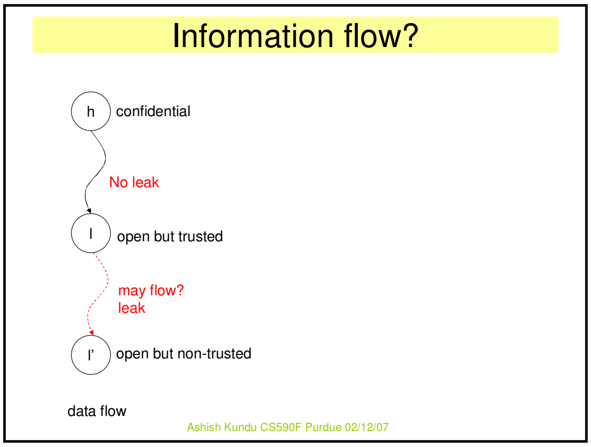
Information flow?



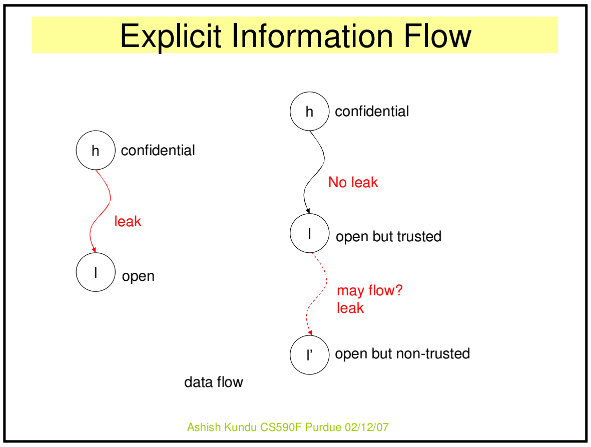
encrypted: h → l

e.g. password sharing

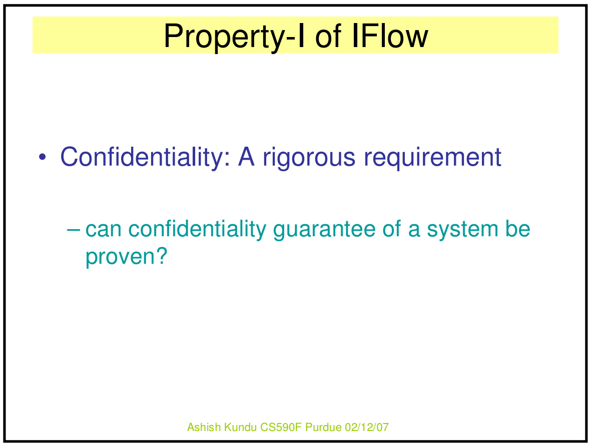
data flow



Information flow?

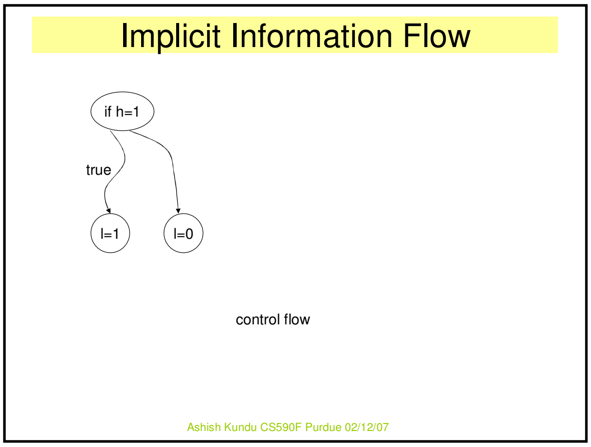


Explicit Information Flow

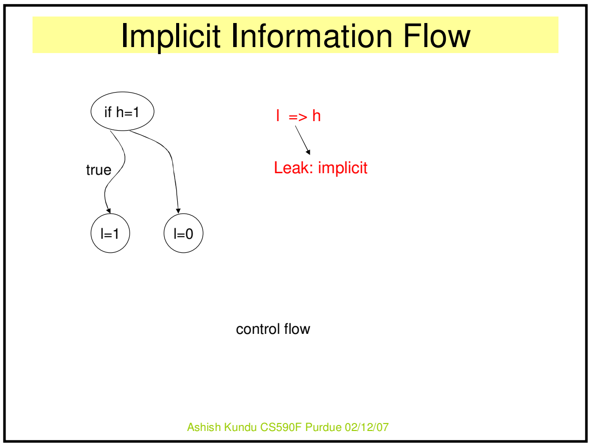


Property-I of IFlow

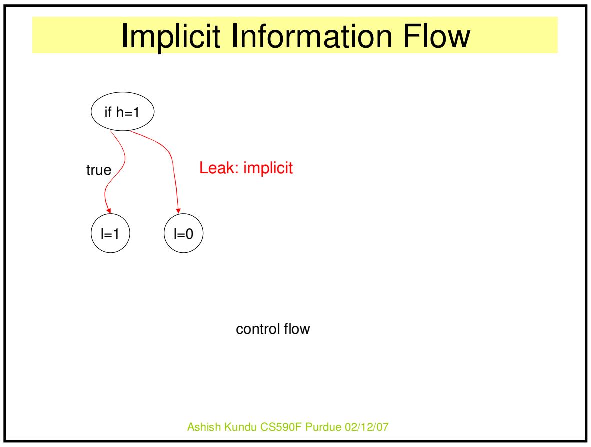
- Confidentiality: A rigorous requirement
  - can confidentiality guarantee of a system be proven?



Implicit Information Flow



Implicit Information Flow



Implicit Information Flow

Property-I of IFlow

- Confidentiality: A rigorous requirement
  - can confidentiality guarantee of a system be proven?
  - can explicit and implicit flows be controlled?
- Relationship with data and control dependency ???

Covert channels

- Implicit flows
  - covert
- Termination channel
  - termination-sensitive confidentiality
- Timing channels
  - subsumes termination channel
- Probabilistic channel
  - PDF of output data
- Resource exhaustion channel
  - memory or disk space: high value for malloc()
- Power channels
  - related: recent work about the age of running system – thus attack vulnerability

Properties of IFlow

- No propagation of <u>high</u> confidential data to <u>low</u> confidential container
- Rigor: On all paths - no leak
  - makes it easy for static-time solutions

Mechanisms

- Access control
  - controls release of information, **not** propogation
  - no control on "how data is used"
- Language-based techniques
  - Runtime: JVM – applets, sandbox
  - Bytecode verifier
    - no control on propagation
- **<u>Type systems</u>**

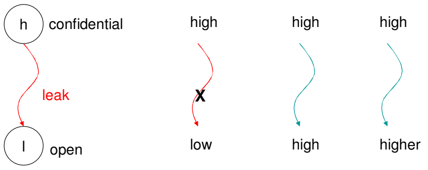
Type systems

- Compositional reasoning
  - incremental construction: from a correct system to a larger and correct system
  - structural induction (will return to this later)
  - objective: correct computation
  - modified objective: correct confidentiality-preserving computation

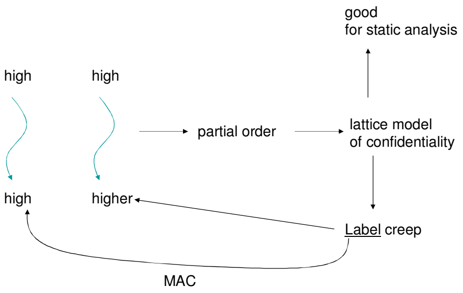
Type systems

- Compositional reasoning
  - incremental construction: from a correct system to a larger and correct system
  - structural induction (will return to this later)
- Objective: correct computation
  - modified objective: correct confidentiality-preserving computation

Explicit Information Flow



Explicit Information Flow



Static Information Flow Control

- Program analysis: Denning and Denning
- Theorem provers
- Type checking

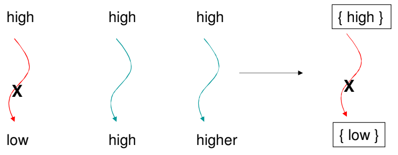
Type checking

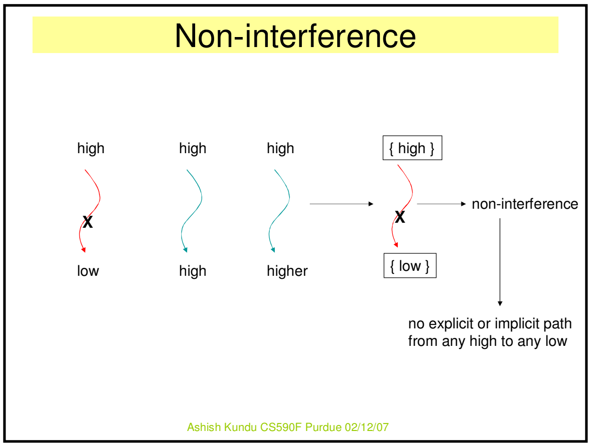
- Security type systems
  - oridinary type: int, char
  - label: static labeling on its confidentiality semantics
- Static type checking detects leaks
  - conservative: so false positive
    - structural induction
  - cannot completely control covert channels
    - semantics – values → Undecidability

Type checking

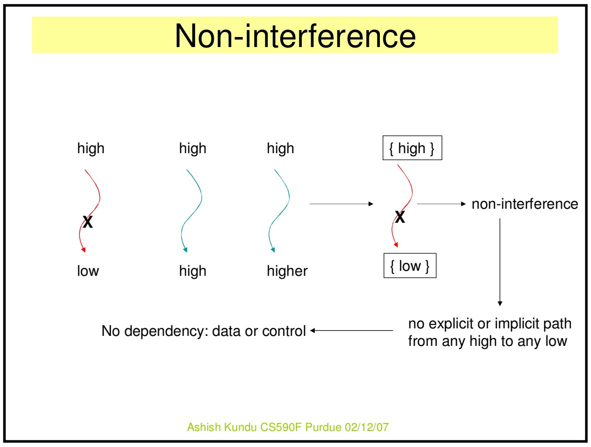
- Security type systems
  - oridinary type: int, char
  - label: static labeling on its confidentiality semantics
- Static type checking detects leaks
  - conservative: so false positive
    - structural induction
  - cannot completely control covert channels
    - semantics – values → Undecidability

Explicit Information Flow



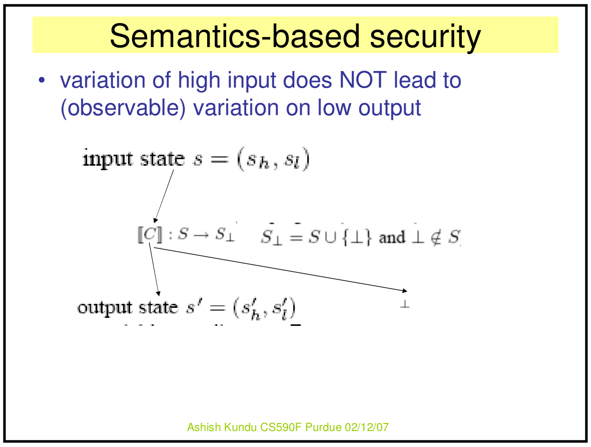


Non-interference



Non-interference

high
high
high
{ high }
X
X
non-interference
low
high
higher
{ low }
No dependency: data or control
no explicit or implicit path from any high to any low



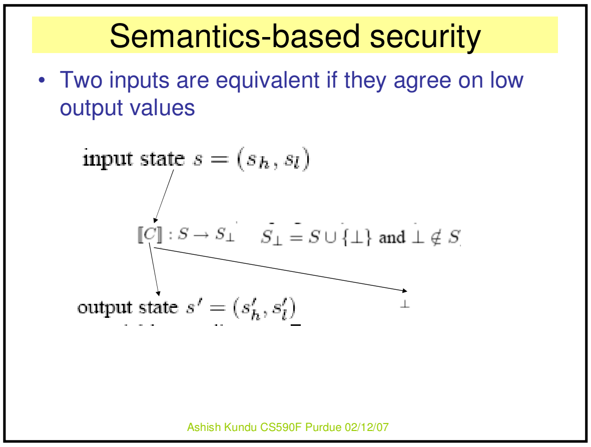
Semantics-based security

- variation of high input does NOT lead to (observable) variation on low output

input state $s = (s_h, s_l)$

$[\![C]\!] : S \to S_\perp$ $\quad S_\perp = S \cup \{\perp\}$ and $\perp \notin S$

output state $s' = (s'_h, s'_l)$ $\quad \perp$



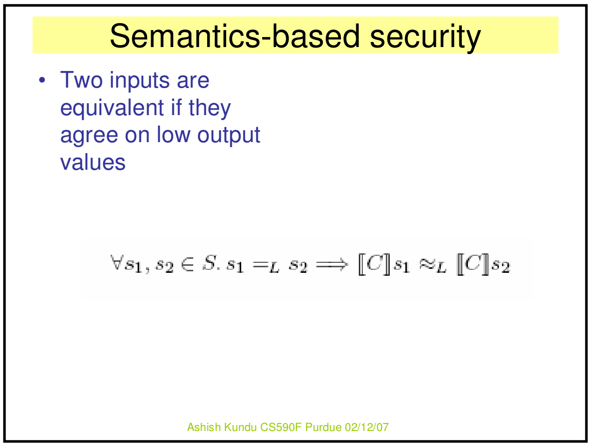
Semantics-based security

- Two inputs are equivalent if they agree on low output values

input state $s = (s_h, s_l)$

$[\![C]\!] : S \to S_\perp$ $\quad S_\perp = S \cup \{\perp\}$ and $\perp \notin S$

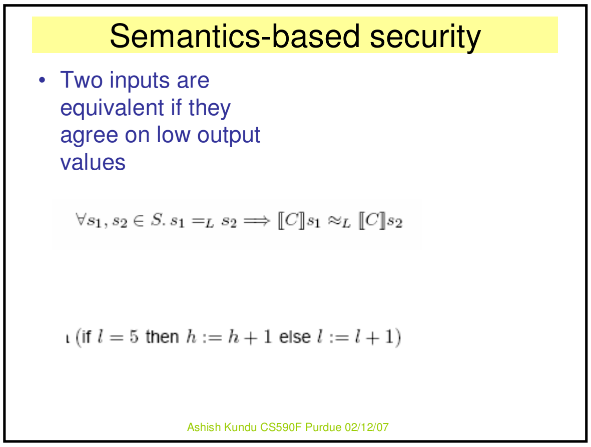
output state $s' = (s'_h, s'_l)$ $\quad \perp$



Semantics-based security

- Two inputs are equivalent if they agree on low output values

$$\forall s_1, s_2 \in S.\, s_1 =_L s_2 \Longrightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$



Semantics-based security

- Two inputs are equivalent if they agree on low output values

$$\forall s_1, s_2 \in S.\, s_1 =_L s_2 \Longrightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$

ι (if $l = 5$ then $h := h + 1$ else $l := l + 1$)

Semantics-based security

- l: = h
- if (h=3) then l:=5 else skip

$$\forall s_1, s_2 \in S.\ s_1 =_L s_2 \Longrightarrow [\![C]\!]s_1 \approx_L [\![C]\!]s_2$$

Security Type System

$$[\mathrm{E1\text{–}2}] \quad \vdash exp : high \qquad \frac{h \notin Vars(exp)}{\vdash exp : low}$$

$$[\mathrm{C1\text{–}3}] \quad [pc] \vdash \mathsf{skip} \qquad [pc] \vdash h := exp \qquad \frac{\vdash exp : low}{[low] \vdash l := exp}$$

$$[\mathrm{C4\text{–}5}] \quad \frac{[pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash C_1; C_2} \qquad \frac{\vdash exp : pc \quad [pc] \vdash C}{[pc] \vdash \mathsf{while}\ exp\ \mathsf{do}\ C}$$

$$[\mathrm{C6\text{–}7}] \quad \frac{\vdash exp : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \mathsf{if}\ exp\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2} \qquad \frac{[high] \vdash C}{[low] \vdash C}$$

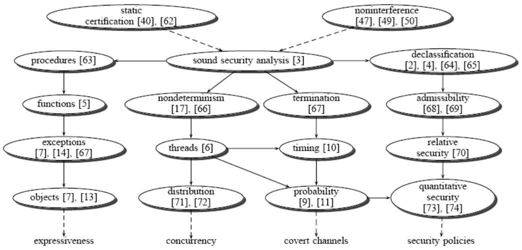
Security Type System

- Restrictive, because it has to be secure in an incremental and compositional manner

Directions

- Expressiveness
- Concurrency
- Covert channels
- Refining security policies

Directions



Expressiveness

- Functions
  - SLam: First-class functions [Heintze et al]
    - non-interference
  - First-class continuations [Zdancewic et al]
    - non-interference
- Exceptions
  - explicit and implicit flows
  - path labeling by Myers
- JFlow by Myers: Java – Jif compiler

Concurrency

- Nondeterminism

Concurrency

- Nondeterminism: possibilistic security condition
  - set of high inputs may not affect set of low outputs
  - dependencies between variables

Concurrency

- Nondeterminism: possibilistic security condition
  - equational security property

$$\forall s \in S.\ [\![HH; C; HH]\!]s \approx [\![C; HH]\!]s$$

Concurrency

- Nondeterminism: possibilistic security condition
  - partial equivalence relations

- PER: symmetric and transitive over a subset of inputs

$$\forall s \in S.\ [\![HH; C; HH]\!]s \approx [\![C; HH]\!]s$$

Concurrency

- Thread concurrency
  - non-atomicity

- Non-interference requirements:
  - no “high” guard in a while loop
  - no if with “high” guard having a while loop in its branch

- termination leak
- timing leak

Concurrency

- Thread concurrency
  - non-atomicity

- Non-interference requirements:
  - no “high” guard in a while loop
  - no if with “high” guard having a while loop in its branch

- termination leak
- timing leak

$$(\text{if } h = 1 \text{ then } C_{long} \text{ else skip});\ l := 1 \ \ \| \ \ l := 0$$

Concurrency

- Thread concurrency
  - non-atomicity
- Scheduler-independent security
  - uniform scheduler [Sabelfield and Sands]
- Type systems: rule out synchronization on “high” data.
  - Sabelfield

Distributed programs

- non-trusted parties
- parties’ concurrency property
- failures
- Secure program partitioning: high and low

Discussion

- Illustrated Security type system : simple yet powerful
  - expressive
  - precise
  - easily extensible to a lattice model of access control
- Organization of the survey addresses
  - all langauge-level factors clearly and precisely
  - illustrates important issues and challenges with simple examples
  - considers both formal approaches and informal aproaches in the light of the
    - hard-ness
    - undecidability of the geneal nature of the problem

Critique

- Presentation very compact: lacking
  - useful illustration and explanation of the concepts and approaches
  - relation between various approaches need to be established
- How to make the approaches such as security type systems part of pragmatic languages
- Needed to address program certification more detailed in a compositional framework

Some Ideas

- Slicing towards proving non-interference
- Use of SSA in checking policy-violations

Some Ideas

- Error Handling: an error violation of integrity policy
  - dual of confidentiality: <high, low> :: <low’, high’>
- Exceptions resulting in termination
  - illegal flow of information?
  - self-healing systems