OREGON PROGRAMMING LANGUAGES SUMMER SCHOOL AT BOSTON UNIVERSITY

Harvard John A. Paulson School of Engineering and Applied Sciences

Language-Based Security

Lecture 3: Information Flow Enforcement

Stephen Chong, Harvard University

Road Map

- Intro
  - Formal Methods for Security
  - Language-Based Security
  - Case Study: Noninterference
- Primer on Computer Security
- Information Flow
  - Semantics
  - Enforcement
  - Beyond confidentiality
- Enforcing Language Abstractions

Enforcement of Information Flow

From Semantics To Enforcement

- We have discussed semantics of information flow
- Very carefully separated from enforcement mechanism
  - I.e., defining our notion of security without how we are going to enforce it
- Let's consider how to enforce noninterference, i.e., control the flow of information in systems

Dimensions of Enforcement

- Enforcement mechanisms differ on granularity and when enforcement occurs
- Granularity:
  - **Coarse grained** mechanisms track information at granularity of *computational containers*
    - Contains both code and data
    - Different granularity of containers, e.g., process, function, block scope, ...
  - **Fine grained** mechanisms track information at level of values/variables
- When does enforcement happen?
  - **Static** mechanisms enforce security before execution
  - **Dynamic** mechanisms enforce security during execution
  - (Hybrid mechanisms use a combination)
- In this lecture, we will look briefly at:
  - Security type system (static fine-grained)
  - Fine-grained information-security monitor (dynamic fine-grained)
  - Coarse-grained information-security monitor (dynamic coarse-grained)

Security-Typed Language

- Type system to enforce (fine-grained) information flow
- Let's see the key ideas in IMP
- Two judgments:

$$\Gamma \vdash e : \tau_\ell \qquad\qquad \Gamma, pc \vdash c$$

Context Γ maps vars to *labeled types*, τ_ℓ

Expression (boolean or arithmetic)

Labeled type
τ ::= int | bool
$\ell \in \Lambda$
Label is upper bound on info that influences the value

Typing of Expressions

$$\Gamma \vdash e : \tau_\ell$$

$$\frac{}{\Gamma \vdash n : \textbf{int}_\bot} \qquad \frac{}{\Gamma \vdash \textbf{true} : \textbf{bool}_\bot} \qquad \frac{}{\Gamma \vdash \textbf{false} : \textbf{bool}_\bot} \qquad \frac{}{\Gamma \vdash x : \Gamma(x)}$$

$$\frac{\Gamma \vdash a_1 : \textbf{int}_{\ell_1} \quad \Gamma \vdash a_2 : \textbf{int}_{\ell_2}}{\Gamma \vdash a_1 + a_2 : \textbf{int}_\ell}\ \ell = \ell_1 \sqcup \ell_2 \qquad \frac{\Gamma \vdash a_1 : \textbf{int}_{\ell_1} \quad \Gamma \vdash a_2 : \textbf{int}_{\ell_1}}{\Gamma \vdash a_1 < a_2 : \textbf{bool}_\ell}\ \ell = \ell_1 \sqcup \ell_2$$

Typing of Commands

$$\Gamma, pc \vdash c$$

Context Γ maps vars to *labeled types*, τ_ℓ

$pc \in \Lambda$

Program counter level

(1) a lower bound on the side effects of c

(2) an upper bound on the info that affects whether this command is executed

Command

Typing of Commands

$$\Gamma, pc \vdash c$$

$$\frac{}{\Gamma, pc \vdash \textbf{skip}}$$

$$\frac{\Gamma \vdash e : \tau_{\ell_e} \quad \ell_e \sqcup pc \sqsubseteq \ell_x}{\Gamma, pc \vdash x := e}\ \Gamma(x) = \tau_{\ell_x}$$

$$\frac{\Gamma, pc \vdash c_1 \quad \Gamma, pc \vdash c_2}{\Gamma, pc \vdash c_1 ; c_2}$$

$$\frac{\Gamma \vdash b : \textbf{bool}_\ell \quad \Gamma, pc \sqcup \ell \vdash c_1 \quad \Gamma, pc \sqcup \ell \vdash c_2}{\Gamma, pc \vdash \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2}$$

$$\frac{\Gamma \vdash b : \textbf{bool}_\ell \quad \Gamma, pc \sqcup \ell \vdash c}{\Gamma, pc \vdash \textbf{while } b \textbf{ do } c}$$

Examples

```
sec := pub + 42;
```

$$\frac{\Gamma \vdash e : \tau_{\ell_e} \quad \ell_e \sqcup pc \sqsubseteq \ell_x}{\Gamma, pc \vdash x := e} \; \Gamma(x) = \tau_{\ell_x}$$

$$\frac{\dfrac{\dfrac{}{\Gamma \vdash \mathsf{pub} : \mathbf{int}_L} \quad \dfrac{}{\Gamma \vdash 42 : \mathbf{int}_L}}{\Gamma \vdash \mathsf{pub} + 42 : \mathbf{int}_L} \quad L \sqcup L \sqsubseteq H}{\Gamma, L \vdash \mathsf{sec} := \mathsf{pub} + 42}$$

```
pub := sec + 42;
```

$$\frac{\dfrac{\dfrac{}{\Gamma \vdash \mathsf{sec} : \mathbf{int}_H} \quad \dfrac{}{\Gamma \vdash 42 : \mathbf{int}_L}}{\Gamma \vdash \mathsf{sec} + 42 : \mathbf{int}_H} \quad {\color{red} H \sqcup L \sqsubseteq L}}{\Gamma, L \vdash \mathsf{pub} := \mathsf{sec} + 42}$$

Examples

```
if (sec < 0)
  sec := -sec
```

$$\frac{\Gamma \vdash b : \textbf{bool}_\ell \quad \Gamma, pc \sqcup \ell \vdash c_1 \quad \Gamma, pc \sqcup \ell \vdash c_2}{\Gamma, pc \vdash \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2}$$

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathsf{sec} < 0 : \textbf{bool}_H} \quad \dfrac{\dfrac{\vdots}{\Gamma \vdash -\mathsf{sec} : \textbf{int}_H} \quad H \sqcup H \sqsubseteq H}{\Gamma, H \vdash \mathsf{sec} := -\mathsf{sec}} \quad \dfrac{}{\Gamma, H \vdash \textbf{skip}}}{\Gamma, L \vdash \textbf{if } \mathsf{sec} < 0 \textbf{ then } \mathsf{sec} := -\mathsf{sec} \textbf{ else skip}}$$

```
if (sec < 0)
  pub := 42
```

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \mathsf{sec} < 0 : \textbf{bool}_H} \quad \dfrac{\dfrac{}{\Gamma \vdash 42 : \textbf{int}_L} \quad H \sqcup L \sqsubseteq L}{\Gamma, H \vdash \mathsf{pub} := 42} \quad \dfrac{}{\Gamma, H \vdash \textbf{skip}}}{\Gamma, L \vdash \textbf{if } \mathsf{sec} < 0 \textbf{ then } \mathsf{pub} := 42 \textbf{ else skip}}$$

Soundness of Type System

- Theorem: For all programs c, if $\Gamma,\bot \vdash c$ then c is noninterfering, i.e.,

  For all σ_1, σ_2, σ'_1, σ'_2, ℓ
  if $\sigma_1 =_\ell \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$
  then $\sigma'_1 =_\ell \sigma'_2$

- Proof:
  - Lots of techniques possible for proving relational properties
  - Direct proof based on induction (on large step operational semantics)
  - Logical relations
  - "Squared" language approach (Due to Pottier & Simonet, 2003)
    - Create a language IMP^2 where one execution of an IMP^2 represents 2 IMP executions
  - …

Another Type System

$$e ::= x \mid n \mid () \mid e_1\ e_2 \mid \lambda x\!:\!\tau, \ell.\, e$$
$$\mid \textsf{input from}\ \ell \mid \textsf{output}\ e\ \textit{to}\ \ell$$
$$\mid \textsf{let}\ x = e_1\ \textsf{in}\ e_2$$
$$\sigma ::= \textbf{unit} \mid \textbf{int} \mid \tau_1 \xrightarrow{pc} \tau_2$$
$$\tau ::= \sigma_\ell$$

Latent effect program counter label
- Is lower bound of side effects of function body
- Is the pc label used to type check function body

Labeled type.
(Label is upper bound on info that influences value of base type σ)

Another Type System

$$e ::= x \mid n \mid () \mid e_1\ e_2 \mid \lambda x : \tau, \ell.\ e$$
$$\mid \textsf{input from}\ \ell \mid \textsf{output}\ e\ \textit{to}\ \ell$$
$$\mid \textsf{let}\ x = e_1\ \textsf{in}\ e_2$$
$$\sigma ::= \textbf{unit} \mid \textbf{int} \mid \tau_1 \xrightarrow{pc} \tau_2$$
$$\tau ::= \sigma_\ell$$

$$\Gamma, pc \vdash e : \tau$$

$$\frac{}{\Gamma, pc \vdash x : \Gamma(x) \sqcup pc} \qquad \frac{}{\Gamma, pc \vdash n : \textbf{int}_{pc}} \qquad \frac{}{\Gamma, pc \vdash () : \textbf{unit}_{pc}}$$

$$\sigma_\ell \sqcup \ell' \triangleq \sigma_{\ell \sqcup \ell'}$$

Another Type System

$$e ::= x \mid n \mid () \mid e_1\ e_2 \mid \lambda x\!:\!\tau, \ell.\, e$$
$$\mid \textsf{input from } \ell \mid \textsf{output } e \textit{ to } \ell$$
$$\mid \textsf{let } x = e_1 \textsf{ in } e_2$$
$$\sigma ::= \textbf{unit} \mid \textbf{int} \mid \tau_1 \xrightarrow{pc} \tau_2$$
$$\tau ::= \sigma_\ell$$

$$\Gamma, pc \vdash e : \tau$$

Info leading to decision to execute the function body

$$\frac{}{\Gamma, pc \vdash x : \Gamma(x) \sqcup pc} \qquad \frac{}{\Gamma, pc \vdash n : \textbf{int}_{pc}} \qquad \frac{}{\Gamma, pc \vdash () : \textbf{unit}_{pc}}$$

$$\frac{\Gamma[x \mapsto \tau], \ell \vdash e : \tau'}{\Gamma, pc \vdash \lambda x\!:\!\tau, \ell.\, e : (\tau \xrightarrow{\ell} \tau')_{pc}} \qquad \frac{\Gamma, pc \vdash e_1 : (\tau \xrightarrow{pc_1} \tau')_{\ell_1} \quad \Gamma, pc \vdash e_2 : \tau \quad \ell_1 \sqcup pc \sqsubseteq pc_1}{\Gamma, pc \vdash e_1\ e_2 : \tau' \sqcup pc}$$

$$\frac{pc \sqsubseteq \ell}{\Gamma, pc \vdash \textsf{input from } \ell : \textbf{int}_{\ell \sqcup pc}}$$

Input is a side effect at level ℓ, so pc must be a lower bound

Another Type System

$$e ::= x \mid n \mid () \mid e_1\ e_2 \mid \lambda x\!:\!\tau, \ell.\, e$$
$$\mid \textsf{input from } \ell \mid \textsf{output } e \textit{ to } \ell$$
$$\mid \textsf{let } x = e_1 \textsf{ in } e_2$$
$$\sigma ::= \textbf{unit} \mid \textbf{int} \mid \tau_1 \xrightarrow{pc} \tau_2$$
$$\tau ::= \sigma_\ell$$

$$\Gamma, pc \vdash e : \tau$$

$$\frac{}{\Gamma, pc \vdash x : \Gamma(x) \sqcup pc} \qquad \frac{}{\Gamma, pc \vdash n : \textbf{int}_{pc}} \qquad \frac{}{\Gamma, pc \vdash () : \textbf{unit}_{pc}}$$

$$\frac{\Gamma[x \mapsto \tau], \ell \vdash e : \tau'}{\Gamma, pc \vdash \lambda x\!:\!\tau, \ell.\, e : (\tau \xrightarrow{\ell} \tau')_{pc}} \qquad \frac{\Gamma, pc \vdash e_1 : (\tau \xrightarrow{pc_1} \tau')_{\ell_1} \quad \Gamma, pc \vdash e_2 : \tau \quad \ell_1 \sqcup pc \sqsubseteq pc_1}{\Gamma, pc \vdash e_1\ e_2 : \tau' \sqcup pc}$$

$$\frac{pc \sqsubseteq \ell}{\Gamma, pc \vdash \textsf{input from } \ell : \textbf{int}_{\ell \sqcup pc}} \qquad \frac{\Gamma, pc \vdash e : \tau \qquad \tau \leq \tau'}{\Gamma, pc \vdash e : \tau'} \qquad \frac{\sigma \leq \sigma' \qquad \ell \sqsubseteq \ell'}{\sigma_\ell \leq \sigma'_{\ell'}}$$

Another Type System

$$e ::= x \mid n \mid () \mid e_1\ e_2 \mid \lambda x\!:\!\tau, \ell.\, e$$
$$\mid \textsf{input from } \ell \mid \textsf{output } e \textit{ to } \ell$$
$$\mid \textsf{let } x = e_1 \textsf{ in } e_2$$
$$\sigma ::= \textbf{unit} \mid \textbf{int} \mid \tau_1 \xrightarrow{pc} \tau_2$$
$$\tau ::= \sigma_\ell$$

$$\Gamma, pc \vdash e : \tau$$

$$\frac{}{\Gamma, pc \vdash x : \Gamma(x) \sqcup pc} \qquad \frac{}{\Gamma, pc \vdash n : \textbf{int}_{pc}} \qquad \frac{}{\Gamma, pc \vdash () : \textbf{unit}_{pc}}$$

$$\frac{\Gamma[x \mapsto \tau], \ell \vdash e : \tau'}{\Gamma, pc \vdash \lambda x\!:\!\tau, \ell.\, e : (\tau \xrightarrow{\ell} \tau')_{pc}} \qquad \frac{\Gamma, pc \vdash e_1 : (\tau \xrightarrow{pc_1} \tau')_{\ell_1} \quad \Gamma, pc \vdash e_2 : \tau \quad \ell_1 \sqcup pc \sqsubseteq pc_1}{\Gamma, pc \vdash e_1\ e_2 : \tau' \sqcup pc}$$

$$\frac{pc \sqsubseteq \ell}{\Gamma, pc \vdash \textsf{input from } \ell : \textbf{int}_{\ell \sqcup pc}} \qquad \frac{\Gamma, pc \vdash e : \tau \qquad \tau \le \tau'}{\Gamma, pc \vdash e : \tau'} \qquad \frac{\sigma \le \sigma' \qquad \ell \sqsubseteq \ell'}{\sigma_\ell \le \sigma'_{\ell'}}$$

$$\frac{\tau_1' \le \tau_1 \qquad \tau_2 \le \tau_2' \qquad pc' \sqsubseteq pc}{\tau_1 \xrightarrow{pc} \tau_2 \ \le\ \tau_1' \xrightarrow{pc'} \tau_2'}$$

• • •

Other Language Features

- Can extend basic ideas of security type system for other language features
  - References (i.e., first-class memory)
  - Exceptions
    - Track information flow associated with normal termination or exceptional termination
  - First-class Labels
  - …

Fine-Grained Dynamic Enforcement

- Dynamic enforcement techniques monitor and restrict execution at runtime
  - Mechanism modifies program behavior! It is an information channel!
  - Need to be aware of what information it reveals by (not) intervening
  - May need to adapt the security condition to account for additional observations

Dynamic Info Flow Tracking

- Flow-Insensitive:

$pc \sqcup \Gamma(\texttt{pub+42}) \sqsubseteq \Gamma(\texttt{sec})$?

$L \sqcup L \sqsubseteq H$ ✓

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

```
sec := pub + 42;
pub := pub + 7;
pub := sec;
pub := 42
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$pc \sqcup \Gamma(\mathrm{pub}+7) \sqsubseteq \Gamma(\mathrm{pub})$?

$L \sqcup L \sqsubseteq L$ ✓

$pc \mapsto L$

$\mathtt{sec} \mapsto H$

$\mathtt{pub} \mapsto L$

```
sec := pub + 42;
pub := pub + 7;
pub := sec;
pub := 42
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$$pc \sqcup \Gamma(\mathtt{sec}) \sqsubseteq \Gamma(\mathtt{pub})$$

$$L \sqcup H \sqsubseteq L \quad ?$$

$$pc \mapsto L$$
$$\mathtt{sec} \mapsto H$$
$$\mathtt{pub} \mapsto L$$

```
sec := pub + 42;
pub := pub + 7;
pub := sec;
pub := 42
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$\Gamma(\texttt{sec>0}) = H$

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

```
if (sec>0) then
  sec := 42
else
  skip;
pub := 0
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$\Gamma(\texttt{sec>0}) = H$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

```
if (sec>0) then
  sec := 42
else
  skip;
pub := 0
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$pc \sqcup \Gamma(42) \sqsubseteq \Gamma(\texttt{sec})$?

$(H \sqcup L) \sqcup L \sqsubseteq H$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

```
if (sec>0) then
  sec := 42
else
  skip;
pub := 0
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$$pc \sqcup \Gamma(0) \sqsubseteq \Gamma(\texttt{pub})$$
$$L \sqcup L \sqsubseteq L$$
?

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

```
if (sec>0) then
  sec := 42
else
  skip;
pub := 0
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$$pc \sqcup \Gamma(\texttt{sec>0}) = H$$

$$pc \mapsto L$$
$$\texttt{sec} \mapsto H$$
$$\texttt{pub} \mapsto L$$

```
if (sec>0) then
  pub := 42
else
  skip
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

$pc \sqcup \Gamma(\texttt{sec>0}) = H$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

```
if (sec>0) then
  pub := 42
else
  skip
```

Dynamic Info Flow Tracking

- Flow-Insensitive:

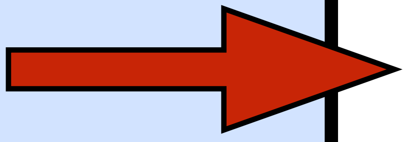
$pc \sqcup \Gamma(42) \sqsubseteq \Gamma(\texttt{pub})$?

$(H \sqcup L) \sqcup L \sqsubseteq L$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$



```
if (sec>0) then
  pub := 42
else
  skip
```

Flow-Sensitive Dynamic

- Natural thing to do is allow the security context to be **flow sensitive**
  - i.e., the mapping from vars to security levels can change during execution
  - (Can do a similar thing with a flow-sensitive type system)
- Accepts more programs!

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\texttt{sec}) = H$

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
x := sec;
x := 0;
output sec to L
```

Flow-Sensitive Dynamic

$$pc \sqcup \Gamma(\texttt{sec}) = H$$

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto H$

```
x := sec;
x := 0;
output sec to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(0) = L$

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto H$

```
x := sec;
x := 0;
output sec to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(0) = L$

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
x := sec;
x := 0;
output sec to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\mathsf{sec}) \sqsubseteq L$

$L \sqcup H \sqsubseteq L$?

$pc \mapsto L$

$\mathsf{sec} \mapsto H$

$\mathsf{pub} \mapsto L$

$\mathsf{x} \mapsto L$

```
x := sec;
x := 0;
output sec to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\texttt{sec>0}) = H$

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
if (sec > 0)
   x := 1
else
   skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\texttt{sec>0}) = H$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
if (sec > 0)
   x := 1
else
   skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(0) = H$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
if (sec > 0)
   x := 1
else
   skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(0) = H$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto H$

```
if (sec > 0)
   x := 1
else
   skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\mathbf{x}) \sqsubseteq L$

$L \sqcup H \sqsubseteq L$?

$pc \mapsto L$

$\mathtt{sec} \mapsto H$

$\mathtt{pub} \mapsto L$

$\mathtt{x} \mapsto H$

```
if (sec > 0)
  x := 1
else
  skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\texttt{sec>0}) = H$

$pc \mapsto L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
if (sec > 0)
  x := 1
else
  skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\texttt{sec>0}) = H$

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
if (sec > 0)
  x := 1
else
  skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \mapsto H \sqcup L$

$\texttt{sec} \mapsto H$

$\texttt{pub} \mapsto L$

$\texttt{x} \mapsto L$

```
if (sec > 0)
  x := 1
else
  skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\mathtt{x}) \sqsubseteq \Gamma(\mathtt{L})$?

$pc \mapsto L$

$\mathtt{sec} \mapsto H$

$\mathtt{pub} \mapsto L$

$\mathtt{x} \mapsto L$

- This is an implicit (aka indirect) flow!
- If we allow it, on some executions we will leak information.
  - So called "half-bit" leak.
  - Can combine 2 "half-bit" leaks to reliably leak a bit!

```
if (sec > 0)
   x := 1
else
   skip;
output x to L
```

Flow-Sensitive Dynamic

$pc \sqcup \Gamma(\mathtt{x}) \sqsubseteq \Gamma(\mathtt{L})$

- This is an implicit (aka indirect) flow!
- If we allow it, on some executions we will leak information.
  - So called “half-bit” leak.
  - Can combine 2 “half-bit” leaks to reliably leak a bit!

```
x := 0; y := 1;
if (sec > 0)
  x := 1
else
  skip;
if (x = 1)
  skip
else
  y := 0
output y to L
```

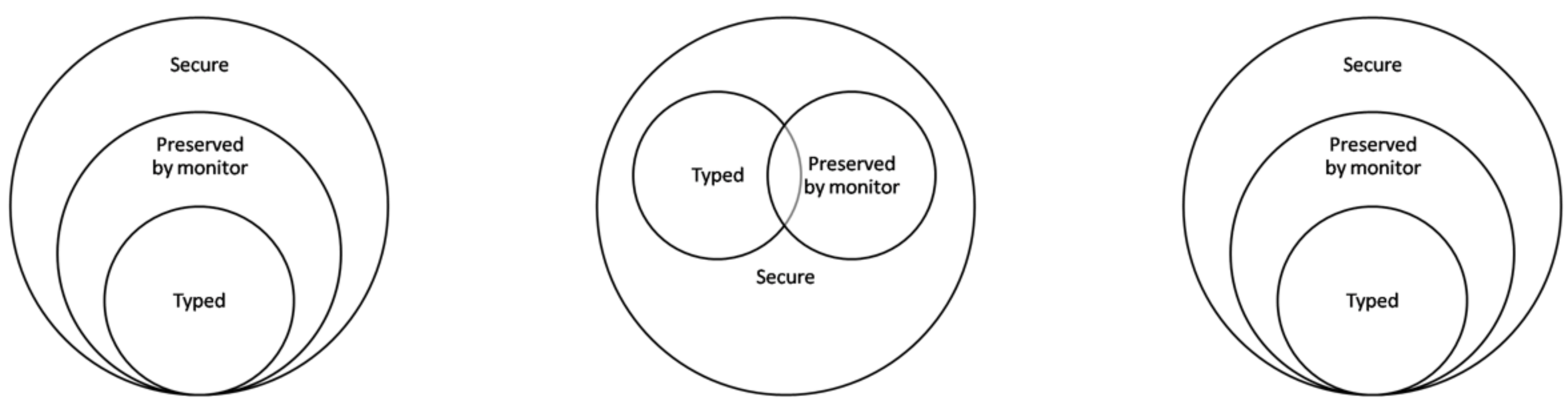
No-Sensitive Upgrade

- Austin and Flanagan (2009)
- Don't raise level of variables when *pc* is high
  - i.e., only raise level of variable x if currently $pc \sqsubseteq \Gamma(x)$
- Some slightly more permissive variations are possible

Dynamic vs Static

- Flow-**insensitive** dynamic tracking can be more precise (for termination-insensitive NI) than flow-insensitive type system
- Flow-**sensitive** dynamic tracking and flow-sensitive type system are incomparable (for termination-insensitive NI)
- **Hybrid systems** combine static and dynamic techniques

Russo & Sabelfeld 2010



(a) Flow-insensitive analysis (b) Flow-sensitive analysis (c) Flow-sensitive analysis, *hybrid* monitors

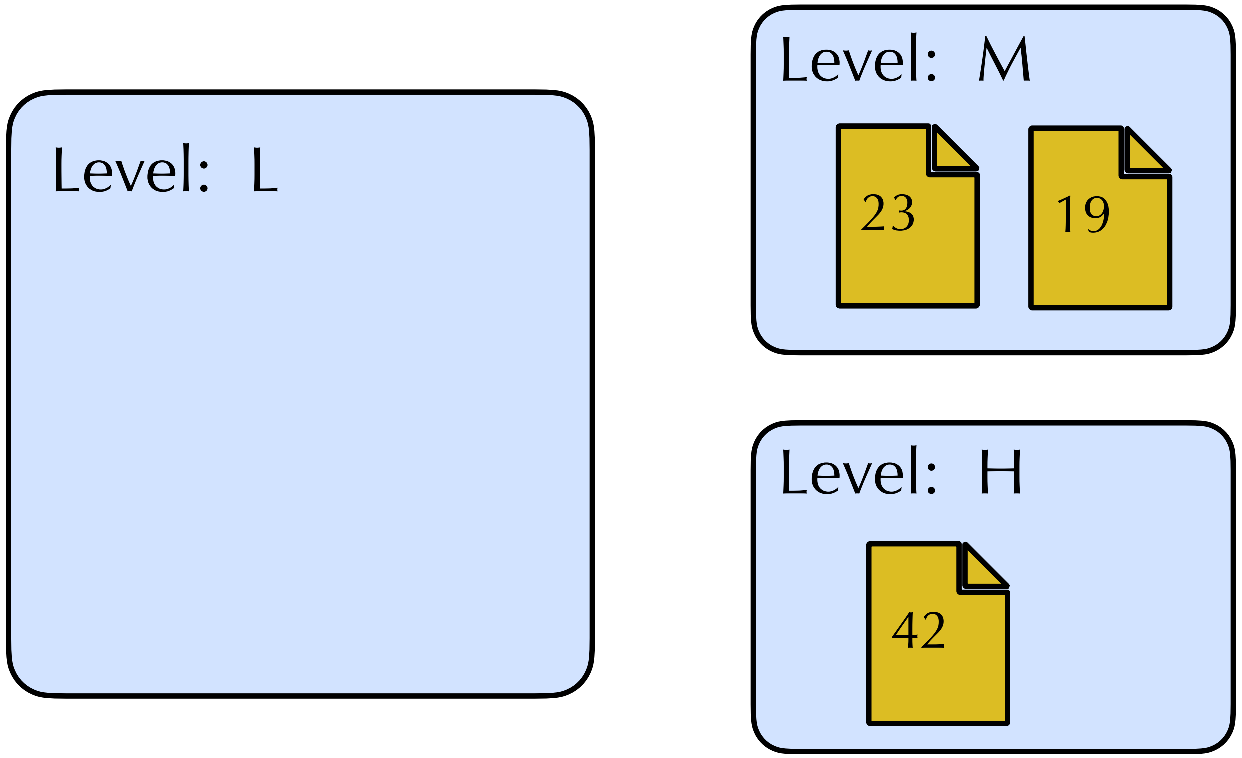
Figure 2. Relation between programs accepted by type systems and monitors

Other Fine-Grained Enforcement Mechanisms

- Dataflow analyses
- Abstract interpretation
- Program dependence graphs/program slicing
- Program rewriting
- Symbolic execution
- Relational program logics
- …

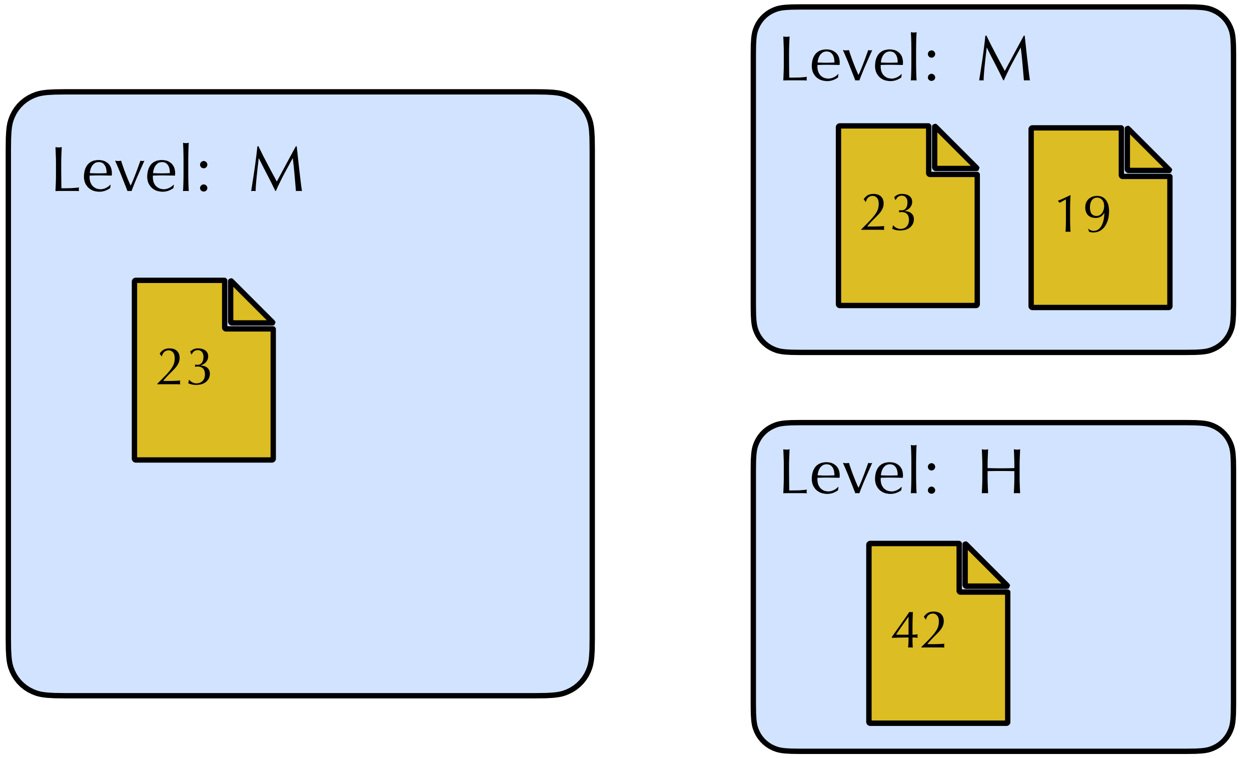
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)



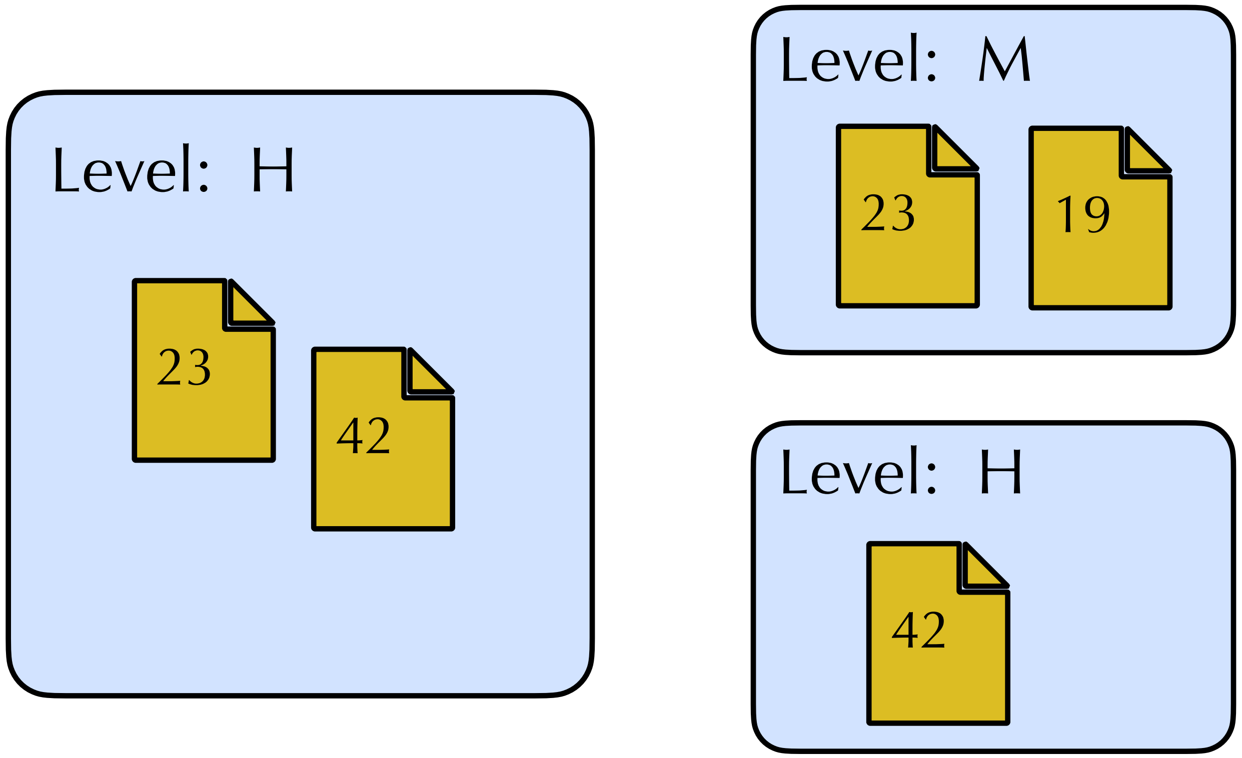
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)



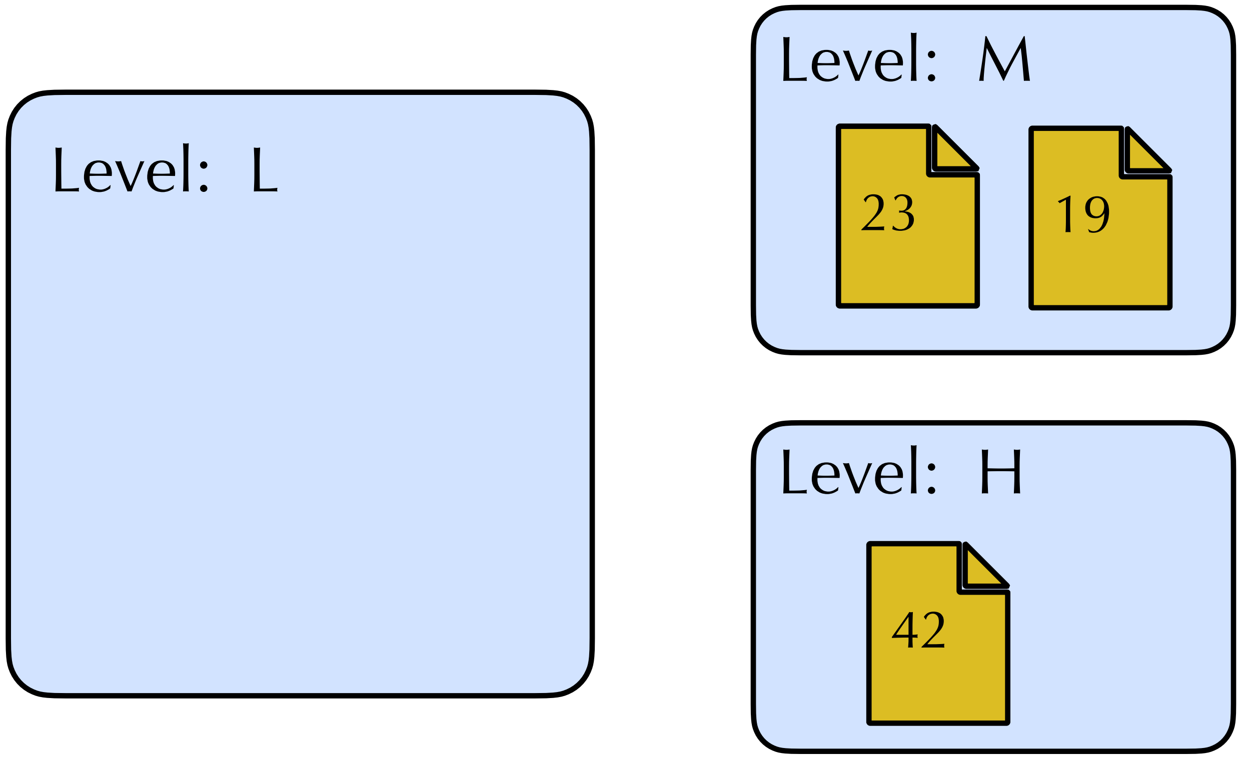
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)



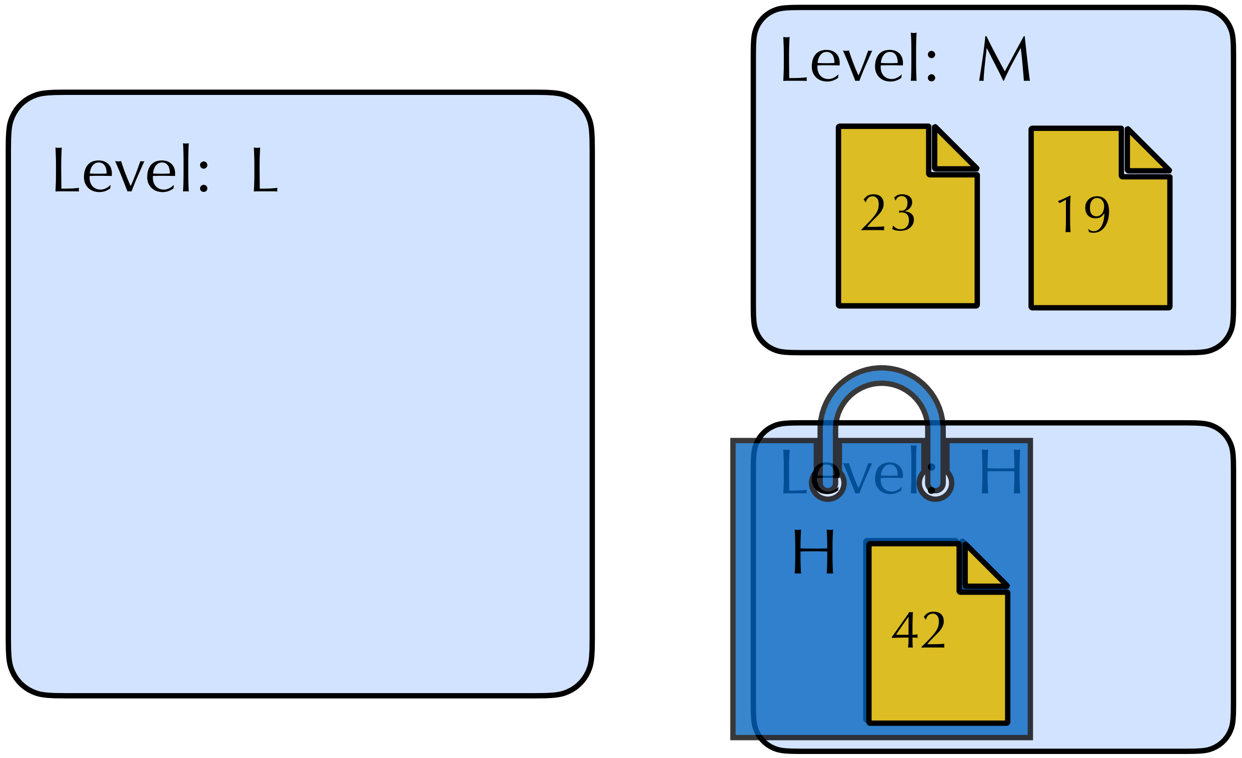
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)



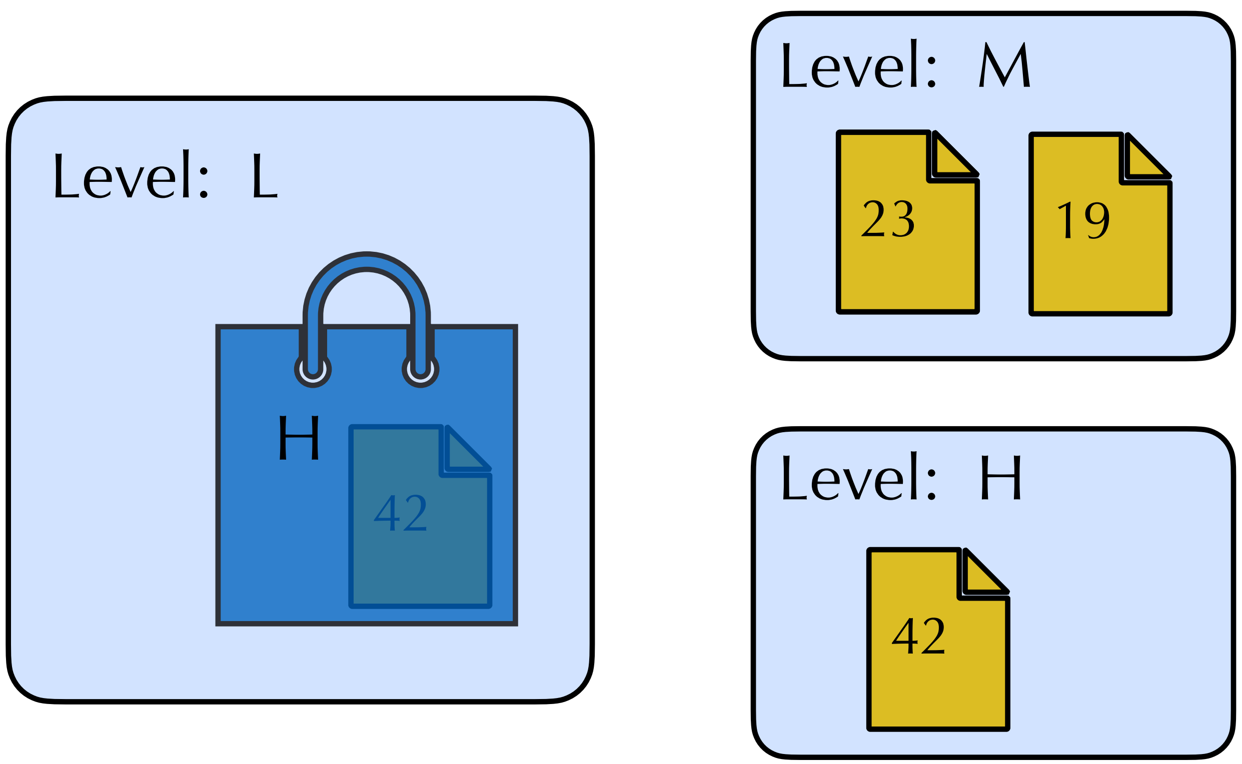
Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)



Coarse-Grain Info Flow Control

- **Computation containers** track what information comes into container
  - Think process, or maybe object
  - Maintain a **high-water mark**: highest security level seen
  - All info in container is treated as potentially tainted with high water mark
- Coarse-grained enforcement is typically dynamic (with maybe some static techniques to enforce the interfaces of the containers)



Selected References

- Volpano, D., G. Smith, and C. Irvine (1996). A sound type system for secure flow analysis. Journal of Computer Security 4(3), 167–187.
- Austin, T. H. and C. Flanagan (2009). Efficient purely-dynamic information flow analysis. In Proceedings of the 2009 Workshop on Programming Languages and Analysis for Security.
- Sabelfeld, A. and A. Russo (2009). From dynamic to static and back: Riding the roller coaster of information- flow control research. In Proceedings of Andrei Ershov International Conference on Perspectives of System Informatics, pp. 352–365.
- Russo, A. and A. Sabelfeld (2010). Dynamic vs. static flow-sensitive security analysis. In Proceedings of the IEEE Computer Security Foundations Symposium.

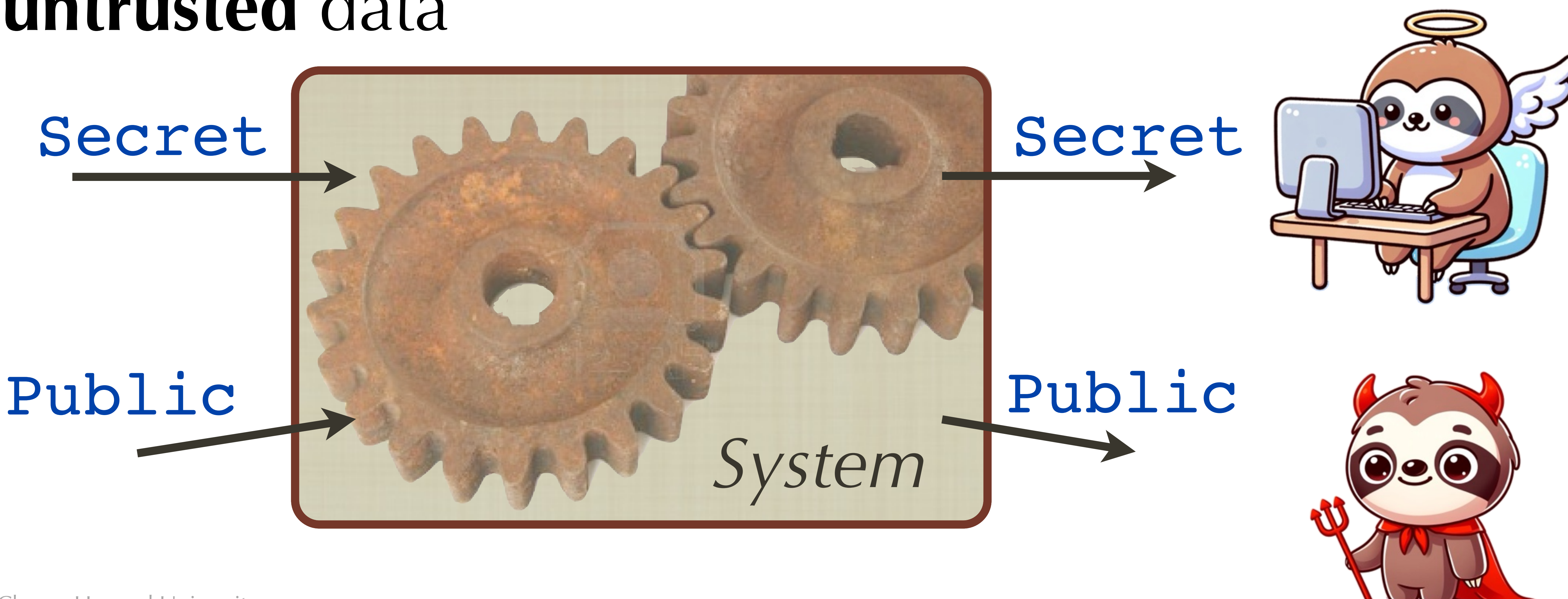
Beyond Confidentiality

Confidentiality and Integrity

- So far, we have considered information flow for confidential information
- We can also think about information flow for integrity

Confidentiality and Integrity

- For confidentiality: we want to restrict flow of **secret** data
- For integrity: we want to restrict flow of **untrusted** data



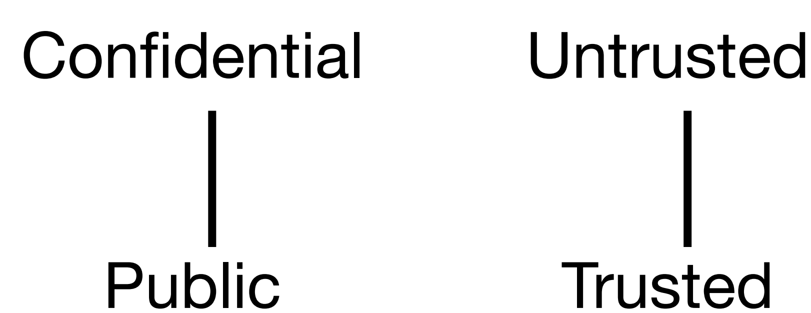
Confidentiality and Integrity

- For confidentiality: we want to restrict flow of **secret** data
- For integrity: we want to restrict flow of **untrusted** data

Noninterference

- The semantic condition is exactly the same!
- The duality between confidentiality and integrity is the direction of "trust" in the lattice

- Definition: Program c is **noninterfering** if:
  For all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2, \ell$
  if $\sigma_1 =_\ell \sigma_2$ and $\langle c,\sigma_1\rangle \Downarrow \sigma'_1$ and $\langle c,\sigma_2\rangle \Downarrow \sigma'_2$
  then $\sigma'_1 =_\ell \sigma'_2$



However...

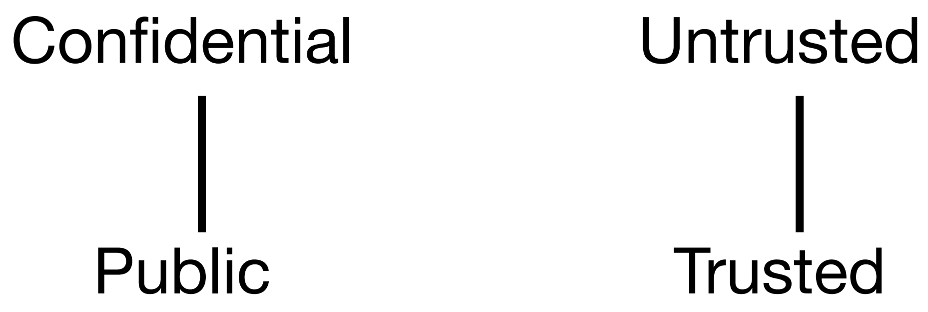
- There are differences between confidentiality and integrity
- Code
  - Many well-principled mechanisms for the integrity of code
    - Code signing
    - Checking of mobile code (bytecode verification, proof-carrying code, type checking, ...)
    - Sandboxing
  - Not so for confidentiality
    - There are impossibility results about the confidentiality of code...

More Differences

- Termination, timing, power consumption, and other side channels
  - Maybe less severe...
    - Do we care if the attacker can affect the acoustic emanations of a CPU?
  - Some covert channel attacks become availability attacks, resource consumption attacks

Reclassification

- The dual of declassification is called **endorsement**
  - Declassification: making information less confidential
  - Endorsement: making information more trusted
- Both move information downwards in the lattice

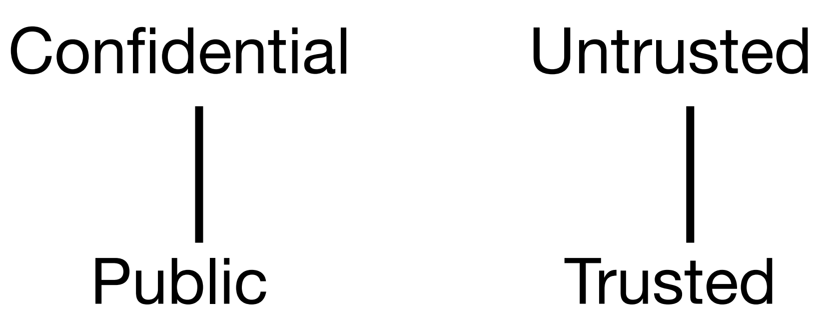


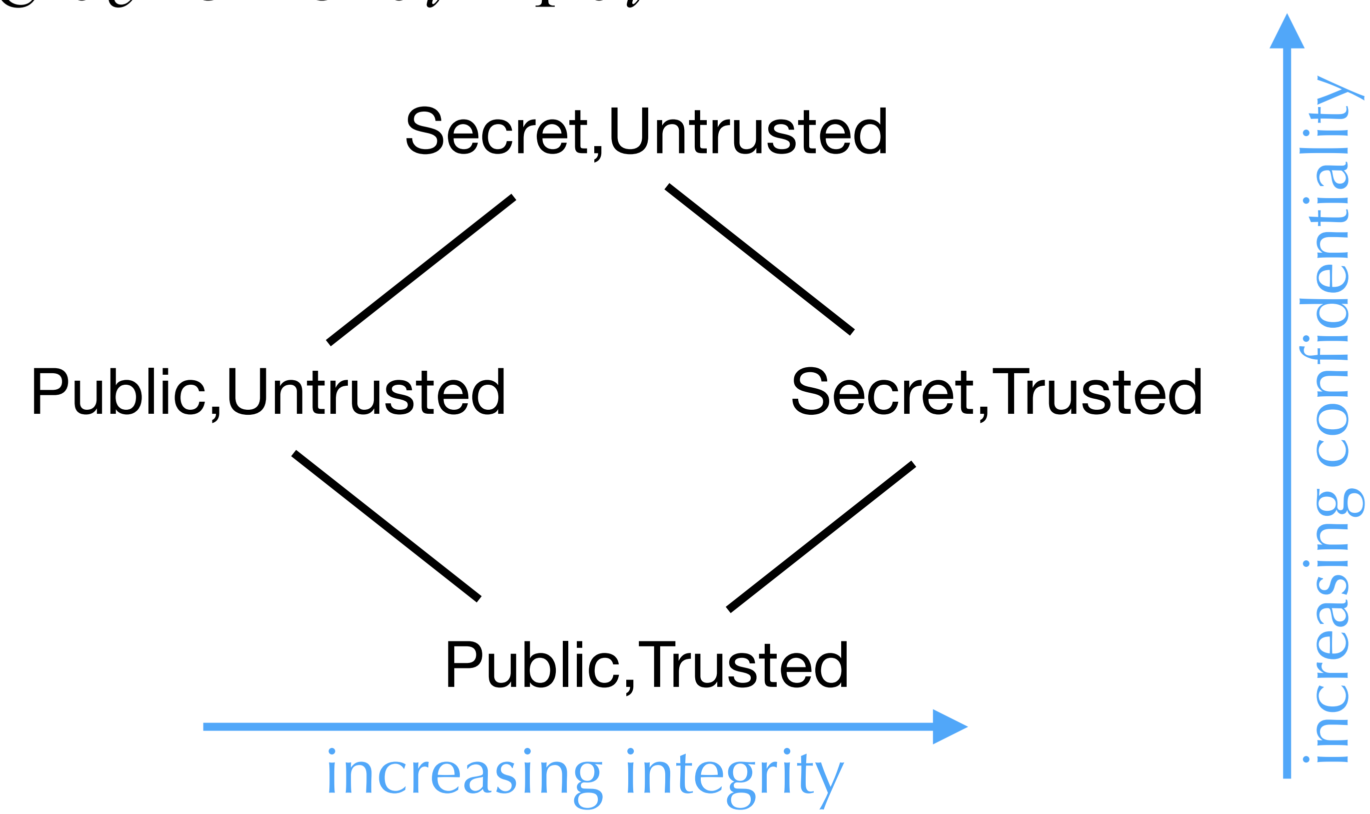
Endorsement

- Aspects of declassification apply to endorsement
  - *What* information is being endorsed?
  - *Who* is responsible for endorsing it? *Who* receives the endorsed information?
  - *Where* in the system (or info-flow lattice) does endorsement happen
  - *When* is information endorsed?
- Quantitative information flow: *how much* information is leaked
  - Contamination vs suppression (Clarkson & Schneider)
  - Contamination = how much untrusted input contaminates trusted output
    - Dual for confidentiality: how much secret input present in public output
  - Suppression = how much trusted input is suppressed in trusted output
    - No confidentiality dual!

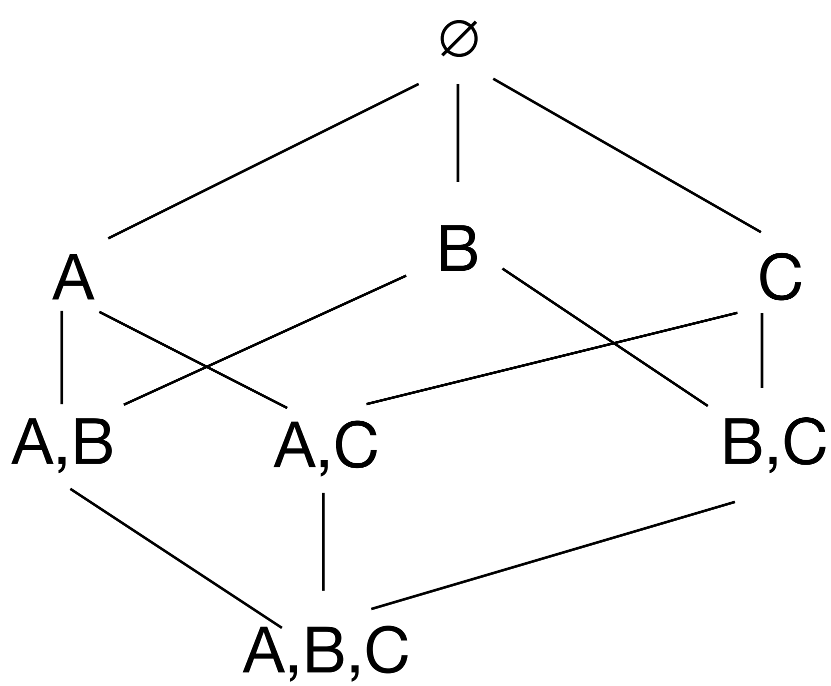
Combining Confidentiality and Integrity

- Given a lattice for confidentiality $(\Lambda_C, \sqsubseteq_C)$ and a lattice for integrity $(\Lambda_I, \sqsubseteq_I)$, we can combine them into a single lattice $(\Lambda, \sqsubseteq)$ where
  - $\Lambda = \Lambda_C \times \Lambda_I = \{ (\ell_c, \ell_i) \mid \ell_c \in \Lambda_C, \ell_i \in \Lambda_I \}$
  - $(\ell_c, \ell_i) \sqsubseteq (\ell_c', \ell_i')$ iff $\ell_c \sqsubseteq_C \ell_c'$ and $\ell_i \sqsubseteq_I \ell_i'$

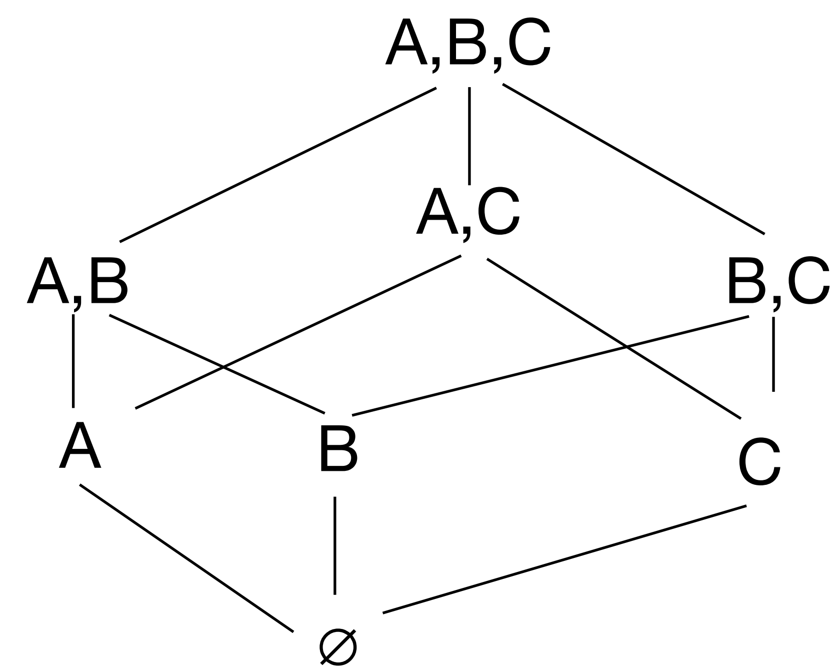




Combining Confidentiality and Integrity



×



Confidentiality Levels
Who can **read** information?
E.g., in A,B, Alice can read it, and Bob can read it (Charlie can not)

Integrity Levels
Who can **write** information?
E.g., in A,B, Alice can write it, and Bob can write it (Charlie can not)

Interaction Between Confidentiality and Integrity

- Consider a program that declassifies some data

```
secret1 := ...;
secret2 := ...;
x := secret1;
pub = declassify(x)
```

- Suppose the attacker can influence which secret is declassified

```
secret1 := ...;
secret2 := ...;
if (low_input) then x := secret1
               else x := secret2
pub = declassify(x)
```

- Attacker can cause the wrong data to be declassified
  - So-called “laundering attack”

Robust Declassification

- Zdancewic and Myers (2001)
- Intuitive idea: an active attacker should not learn more than a passive attacker
  - Active attacker: providing low-integrity inputs
  - Passive attacker: just observing
- This implies that the data to declassify, and the decision to declassify it, should be high integrity

Typing Rule for Robust Declassification

- Rule for assignment

$$\frac{\Gamma \vdash e : \tau_{\ell_e} \quad \ell_e \sqcup pc \sqsubseteq \ell_x}{\Gamma, pc \vdash x := e} \; \Gamma(x) = \tau_{\ell_x}$$

- Equivalent rule for assignment

$$\frac{\Gamma \vdash e : \tau_{\ell_e} \quad pc \sqsubseteq \ell_x \quad \ell_e \sqsubseteq \ell_x}{\Gamma, pc \vdash x := e} \; \Gamma(x) = \tau_{\ell_x}$$

Typing Rule for Robust Declassification

- Equivalent rule for assignment

$$\frac{\Gamma \vdash e : \tau_{\ell_e} \quad pc \sqsubseteq \ell_x \quad \ell_e \sqsubseteq \ell_x}{\Gamma, pc \vdash x := e} \; \Gamma(x) = \tau_{\ell_x}$$

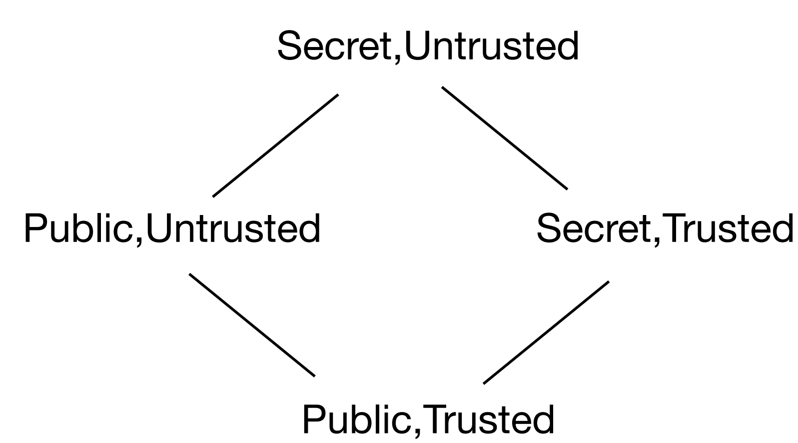
Data to declassify is trusted

- Rule for declassification

$$\frac{\Gamma \vdash e : \tau_{\ell_{from}} \quad pc \sqsubseteq \ell_{to} \quad pc \sqsubseteq (\mathsf{Secret}, \mathsf{Trusted}) \quad \ell_{from} \sqsubseteq (\mathsf{Secret}, \mathsf{Trusted}) \quad \mathsf{integOf}(\ell_{from}) = \mathsf{integOf}(\ell_{to})}{\Gamma, pc \vdash x := \mathsf{declassify}(e)} \; \Gamma(x) = \tau_{\ell_{to}}$$

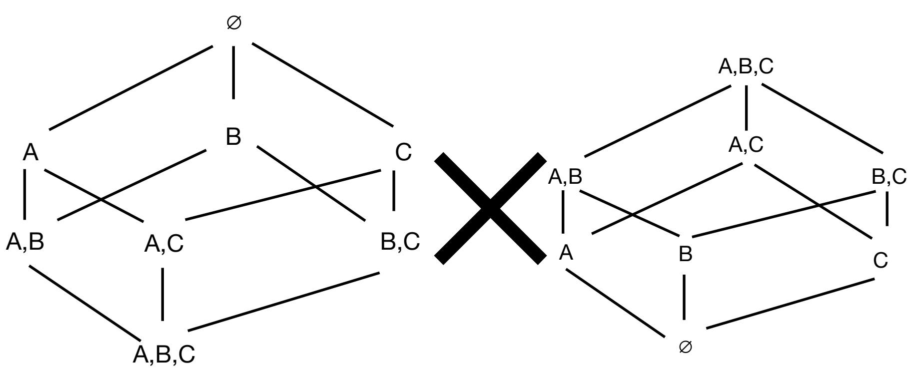
Decision to declassify is trusted

It is declassification only, not endorsement

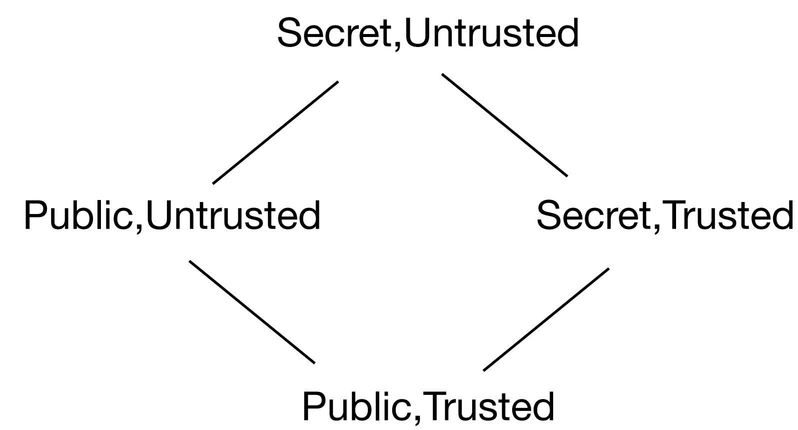


Typing Rule for Robust Declassification

$$\frac{\Gamma \vdash e : \tau_{\ell_{from}} \quad pc \sqsubseteq \ell_{to} \quad pc \sqsubseteq (\mathsf{Secret}, \mathsf{Trusted}) \quad \ell_{from} \sqsubseteq (\mathsf{Secret}, \mathsf{Trusted}) \quad \mathsf{integOf}(\ell_{from}) = \mathsf{integOf}(\ell_{to})}{\Gamma, pc \vdash x := \mathsf{declassify}(e)} \; \Gamma(x) = \tau_{\ell_{to}}$$



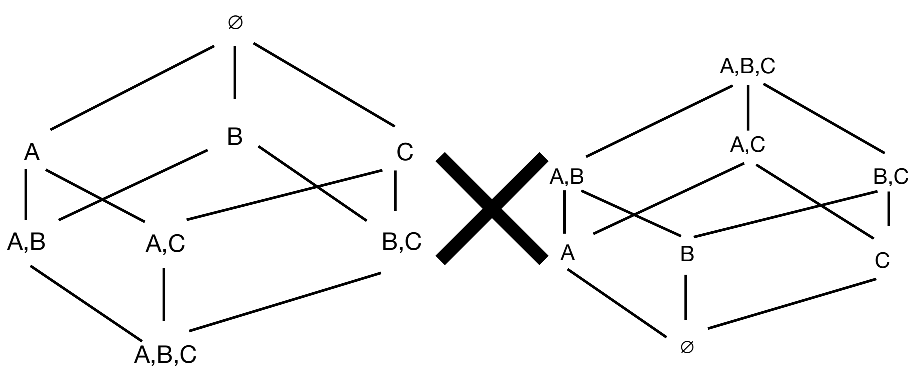
Reader Set Writer Set



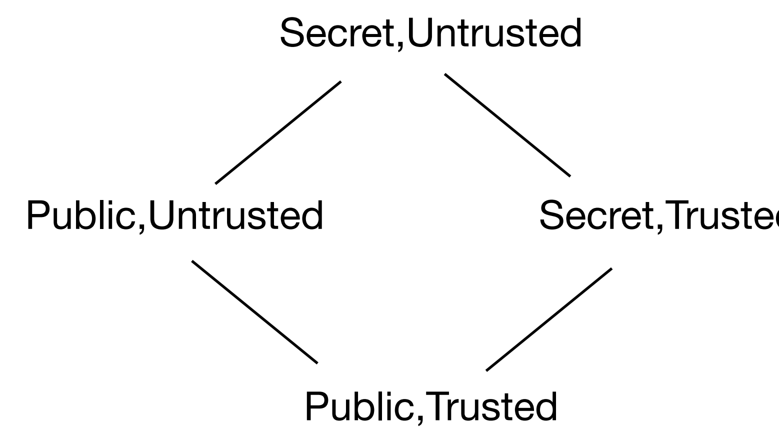
- Intuition: for any principal p, if the declassification lets p read the data, p should not have influenced it
  - ∀p. p ∈ readers(ℓ_{to}) – readers(ℓ_{from}) ⇒ p ∉ writers(ℓ_{from})
  - ∀p. p ∈ readers(ℓ_{to}) ⇒ p ∈ readers(ℓ_{from}) or p ∉ writers(ℓ_{from})
  - readers(ℓ_{from}) ⊇ readers(ℓ_{to}) ∩ writers(ℓ_{from})
  - readers(ℓ_{from}) $\sqsubseteq_C$ readers(ℓ_{to}) ⊔ writers(ℓ_{from})
  - $\ell_{from} \sqsubseteq \ell_{to} \sqcup$ writersToReaders(ℓ_{from})

Typing Rule for Robust Declassification

$$\frac{\Gamma \vdash e : \tau_{\ell_{from}} \quad pc \sqsubseteq \ell_{to} \quad pc \sqsubseteq (\text{Secret}, \text{Trusted}) \quad \ell_{from} \sqsubseteq (\text{Secret}, \text{Trusted}) \quad \text{integOf}(\ell_{from}) = \text{integOf}(\ell_{to})}{\Gamma, pc \vdash x := \text{declassify}(e)} \; \Gamma(x) = \tau_{\ell_{to}}$$



Reader Set　　Writer Set



$$\frac{\Gamma \vdash e : \tau_{\ell_{from}} \quad pc \sqsubseteq \ell_{to} \quad \text{integOf}(\ell_{from}) = \text{integOf}(\ell_{to}) \qquad \ell_{from} \sqsubseteq \ell_{to} \sqcup \text{writersToReaders}(\ell_{from}) \quad \ell_{from} \sqsubseteq \ell_{to} \sqcup \text{writersToReaders}(pc)}{\Gamma, pc \vdash x := \text{declassify}(e)} \; \Gamma(x) = \tau_{\ell_{to}}$$

Data to declassify is trusted

Decision to declassify is trusted

What About Endorsement?

- Equivalent of robust declassification for integrity is **transparent endorsement** (Cecchetti et al., 2017)
- Intuitively: data and decision to endorse should be public
- Nonmalleable info flow =
  robust declassification
  + transparent endorsement

Dependency

- At its core, noninterference is about **(in)dependency**
- Techniques for noninterference are also good for dependency
- E.g.,
  - Binding-time analysis, slicing, ... (Abadi et al. 1999)
  - Tracking and restricting errors in computation (Sampson et al. 2011)

Selected References

- Zdancewic, S. and A. C. Myers (2001, June). Robust declassification. In Proceedings of the 14th IEEE Computer Security Foundations Workshop, Cape Breton, Nova Scotia, Canada, pp. 15–23. IEEE Computer Society.
- Abadi, M., A. Banerjee, N. Heintze, and J. G. Riecke (1999). A core calculus of dependency. In Conference Record of the Twenty-Sixth Annual ACM Symposium on Principles of Programming Languages, New York, NY, USA, pp. 147–160. ACM Press.
- Sampson, A., W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman (2011). Enerj: approxi- mate data types for safe and general low-power computation. In Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '11, New York, NY, USA, pp. 164–174. Association for Computing Machinery.
- Cecchetti, E., A. C. Myers, and O. Arden (2017). Nonmalleable information flow control. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17, New York, NY, USA, pp. 1875–1891. Association for Computing Machinery.