

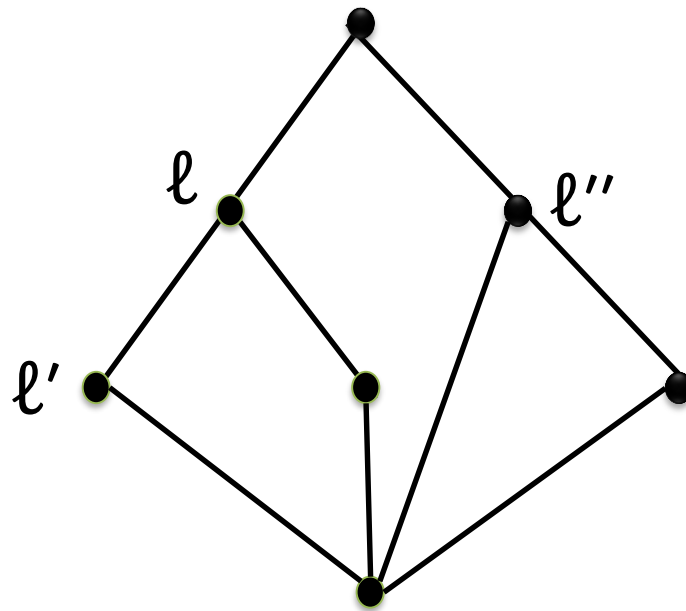
CS 5430

Dynamic Information-Flow Control

Elisavet Kozyri

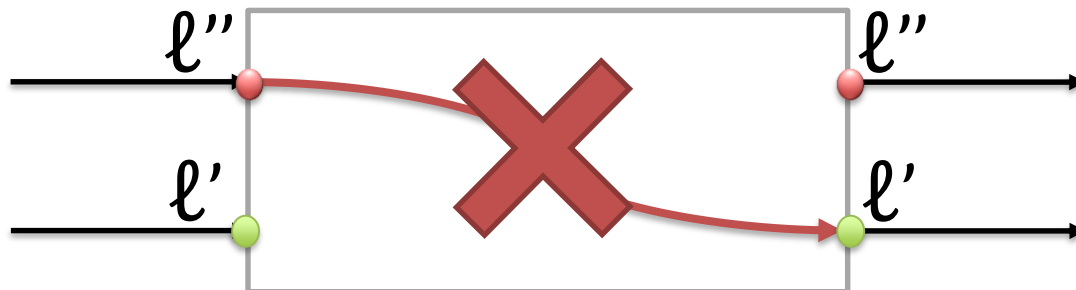
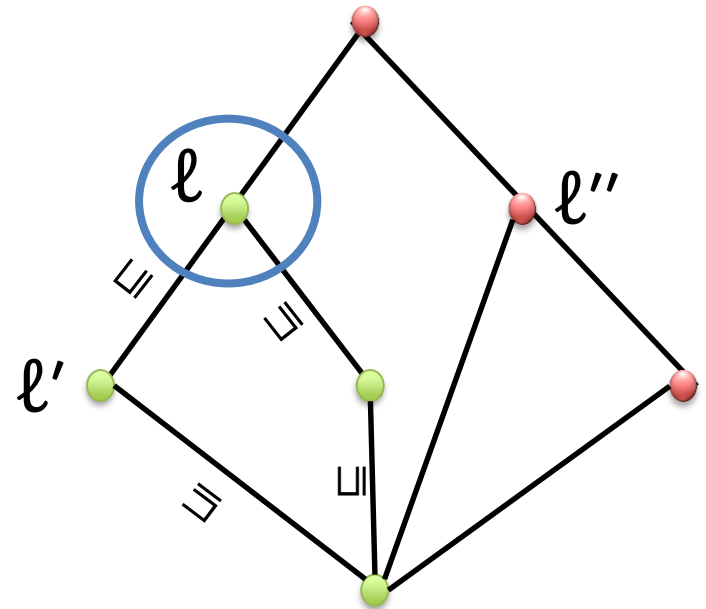
Spring 2019

Lattice of information flow labels



Noninterference $\forall \ell$

- Green labels are considered “low” with respect to ℓ .
- Red labels are considered “high” with respect to ℓ .
- Values tagged with red labels should not flow to values tagged with green labels.



Review: Static type system

$$\frac{\Gamma \vdash \mathbf{e} : \ell \quad \ell \sqcup ctx \sqsubseteq \Gamma(\mathbf{x})}{\Gamma, ctx \vdash \mathbf{x} := \mathbf{e}}$$

$$\frac{\Gamma \vdash \mathbf{e} : \ell \quad \Gamma, \ell \sqcup ctx \vdash \mathbf{c1} \quad \Gamma, \ell \sqcup ctx \vdash \mathbf{c2}}{\Gamma, ctx \vdash \mathbf{if\ e\ then\ c1\ else\ c2}}$$

$$\frac{\Gamma \vdash \mathbf{e} : \ell \quad \Gamma, \ell \sqcup ctx \vdash \mathbf{c}}{\Gamma, ctx \vdash \mathbf{while\ e\ do\ c}}$$

$$\frac{\Gamma, ctx \vdash \mathbf{c1} \quad \Gamma, ctx \vdash \mathbf{c2}}{\Gamma, ctx \vdash \mathbf{c1 ; c2}}$$

Soundness of type system

- Noninterference:
 - $\forall \ell, M_1, M_2: M_1 =_{\ell} M_2 \Rightarrow \mathbf{c}(M_1) =_{\ell} \mathbf{c}(M_2)$
 - where $M_1 =_{\ell} M_2$ denotes equality on all variables tagged with $\ell' \sqsubseteq \ell$, and
 - $\mathbf{c}(M_1) =_{\ell} \mathbf{c}(M_2)$ denotes equality on all outputs tagged with $\ell' \sqsubseteq \ell$.
- $\Gamma, ctx \vdash \mathbf{c}$ implies that \mathbf{c} satisfies NI
- The same type system can enforce noninterference for labels from an arbitrary lattice, for either confidentiality or integrity!

Limitations of the type system



This type system is conservative.

It has false positives:

- There are programs that satisfy noninterference, but they are not type correct.
- Example with $\Gamma(\mathbf{h}) = H$ and $\Gamma(\mathbf{l}) = L$:

if 0=0 then l:=2 else l:=h

Can we build a mechanism with fewer false positives?

Dynamic mechanisms: decrease false positives over static mechanisms through the use of run-time information.

From static to dynamic mechanisms

A dynamic mechanism checks/deduces labels along the execution:

- When an assignment $\mathbf{x} := \mathbf{e}$ is executed,
 - either check whether $\Gamma(\mathbf{e}) \sqcup ctx \sqsubseteq \Gamma(\mathbf{x})$ holds,
 - The execution of a program is blocked when a check fails.
 - or deduce $\Gamma(\mathbf{x})$ such that $\Gamma(\mathbf{e}) \sqcup ctx \sqsubseteq \Gamma(\mathbf{x})$ holds.
- When execution enters a conditional command, the mechanism augments ctx with the label of the guard.
- When execution exits a conditional command, ctx is restored.

A dynamic mechanism: Example

Assume a dynamic enforcement mechanism with fixed Γ , where $\Gamma(\mathbf{h}) = H$ and $\Gamma(\mathbf{1}) = L$.

if $\mathbf{h}=0$ then $\mathbf{h}:=2$ else $\mathbf{h}:=3$; $\mathbf{1}:=\mathbf{h}$

$ctx = L$

$ctx = H$

Check: 

$ctx \sqcup \Gamma(2) \sqsubseteq \Gamma(\mathbf{h})$

$ctx = L$

Check: 

$ctx \sqcup \Gamma(\mathbf{h}) \not\sqsubseteq \Gamma(\mathbf{1})$


Execution blocks!

Comparing static to dynamic

Assume a dynamic enforcement mechanism with fixed Γ , where $\Gamma(\mathbf{h}) = H$ and $\Gamma(\mathbf{1}) = L$.

$\text{if } 0=0 \text{ then } 1:=2 \text{ else } 1:=h$

$\text{ctx} = L$ $\text{ctx} = L$ $\text{ctx} = L$

Check: 

$\text{ctx} \sqcup \Gamma(2) \sqsubseteq \Gamma(\mathbf{1})$

Comparing static to dynamic

- So, under dynamic analysis command
if 0=0 then 1:=2 else 1:=h
- is always executed to completion,
- because dynamic check $\Gamma(2) \sqcup \Gamma(0=0) \sqsubseteq \Gamma(1)$ always succeeds,
- and because branch **1:=h** is never taken.
- The static type system rejects this program before execution, even though the program is secure!

Accepting some executions

Assume a dynamic enforcement mechanism with fixed Γ , where $\Gamma(\mathbf{h}) = H$, $\Gamma(\mathbf{1}) = L$.

Execution blocks!

Check: 

$ctx \sqcup \Gamma(\mathbf{h}) \sqsubseteq \Gamma(\mathbf{1})$

$ctx = L$

$ctx = L$



if $1 < 2$ **then** $1 := 0$ **else** $1 := h$



$ctx = L$

$ctx = L$

$ctx = L$

Check: 

$ctx \sqcup \Gamma(\mathbf{0}) \sqsubseteq \Gamma(\mathbf{1})$

Accepting some executions

- So, for program

if $1 < 2$ then $1 := 0$ else $1 := h$

- If $1 < 2$ holds, then command is executed to termination, because $\Gamma(0) \sqcup \Gamma(1 < 2) \sqsubseteq \Gamma(1)$ succeeds.
- If $1 < 2$ does not hold, then command is blocked before executing $1 := h$, because $\Gamma(h) \sqcup \Gamma(1 < 2) \sqsubseteq \Gamma(1)$ does not succeed.
- Is this program accepted by the static type system?
 - The static type system rejects this program before execution.
 - So, all executions of this program are rejected.

A dynamic mechanism can be more permissive than a static mechanism.

**Another way to increase permissiveness:
use flow-sensitive labels.**

From fixed labels to flow-sensitive labels

- A flow-sensitive label on a variable can change during the analysis of the program.
- Flow-sensitive labels can be used both in a static or dynamic mechanism.

From fixed labels to flow-sensitive labels

$$x := h; \quad x := 0; \quad l := x$$

- Assume $\Gamma(\mathbf{h}) = H$ and $\Gamma(\mathbf{l}) = L$.
- Is this program safe?
- If $\Gamma(x)$ is fixed to H , then the program is rejected, because the analysis of $l := x$ fails.
- If $\Gamma(x)$ is flow-sensitive, then
 - $\Gamma(x)$ becomes H after $x := h$,
 - $\Gamma(x)$ becomes L after $x := 0$, and
 - the analysis of $l := x$ succeeds.
- So, flow-sensitive labels can enhance permissiveness even further.

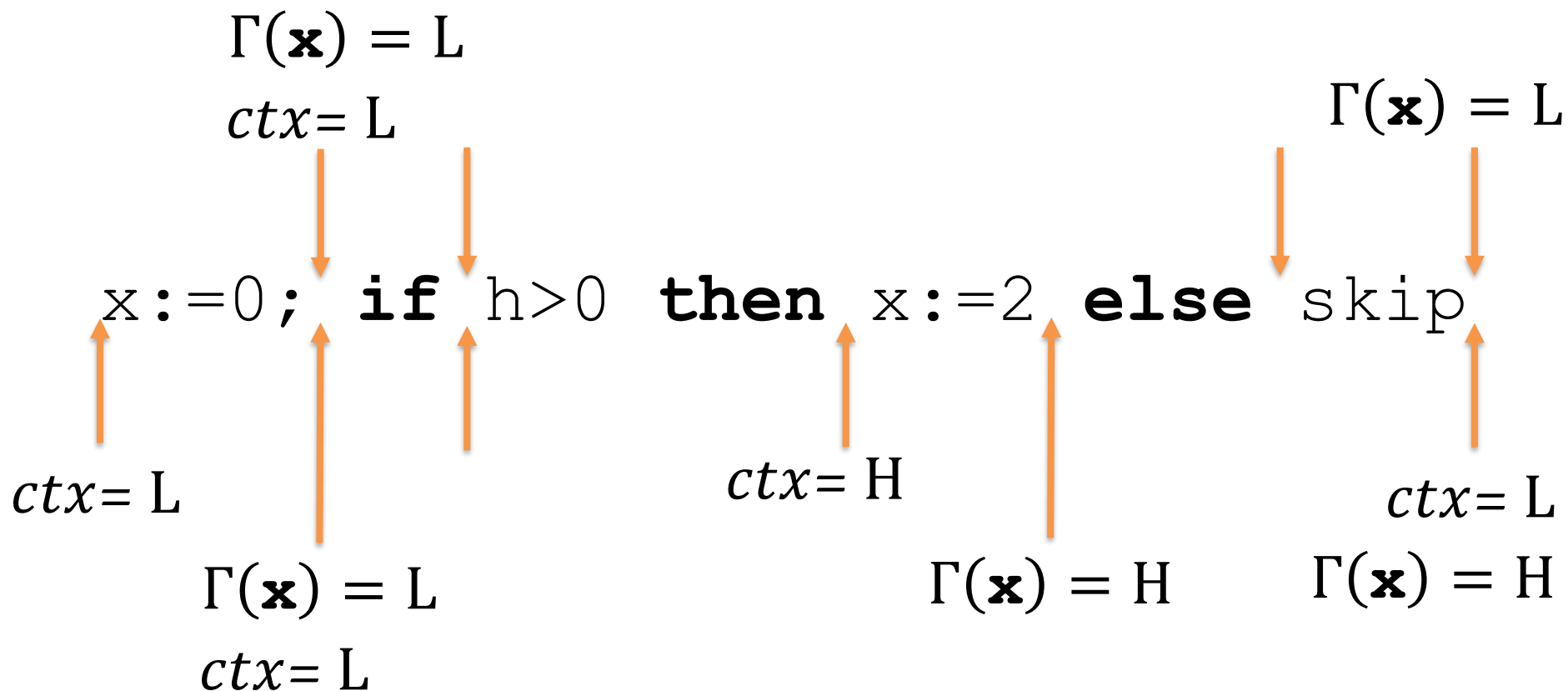
**Combine dynamic mechanisms with
flow-sensitive labels**

Purely dynamic flow-sensitive mechanism

- Analyze only code that is being executed.
- A purely dynamic flow-sensitive mechanism will either be more conservative than a static flow-sensitive mechanism or unsound. [Russo & Sabelfeld, 2010]

Example

Assume fixed $\Gamma(\mathbf{h}) = H$ and flow-sensitive $\Gamma(\mathbf{x})$.



Example

So, for command

$x := 0; \text{ if } h > 0 \text{ then } x := 2 \text{ else skip}$

- If $h > 0$ holds, then after $x := 2$, $\Gamma(\mathbf{x})$ becomes H.
- If $h > 0$ does not hold, then $\Gamma(\mathbf{x})$ remains L.
 - This label is not sound!
- Problem: Even though \mathbf{h} flows to \mathbf{x} , \mathbf{x} is tagged with H only when $\mathbf{h} > 0$; \mathbf{x} is tagged with L when $\mathbf{h} \not> 0$.

How can we recover soundness?

1st solution

- Make purely dynamic flow-sensitive mechanism more conservative:
 - Block execution before entering conditional commands with high guards.
- For previous example:
`x:=0; if h>0 then x:=2 else skip`
All execution would stop after `x:=0`.

2nd solution: Multi-execution

- Execute the program as many times as the labels in the lattice.
- For the execution that corresponds to label ℓ , replace all initial values of variables initially tagged with ℓ' with dummy values, if $\ell' \sqsubseteq \ell$ does not hold.

2nd solution: Multi-execution

```
x := 0; if h > 0 then x := 2 else skip
```

Consider execution of this program with initialization $h=3$.

- H version of the execution will have initialization $h=3$.
- L version of the execution will have initialization $h=0$ (dummy value).
 - Final value of x is 0.

Consider execution of this program with initialization $h=-1$.

- H version of the execution will have initialization $h=-1$.
- L version of the execution will have initialization $h=0$ (dummy value).
 - Final value of x is 0.
- So, there is no flow from high h to low x .

3rd solution: Use on-the-fly static analysis

- An on-the-fly static analysis can update the labels of target variables in untaken branches to capture implicit flow.
- So, the mechanism is no longer purely dynamic.


Use on-the-fly static analysis to capture implicit flow

Problem: **x** was tagged with H only when **h>0** was true, even though **h** always flows to **x**.

Goal: **x** should be tagged with H at every execution.

x := 0 ;

if **h** > 0 then **x** := 1 else skip

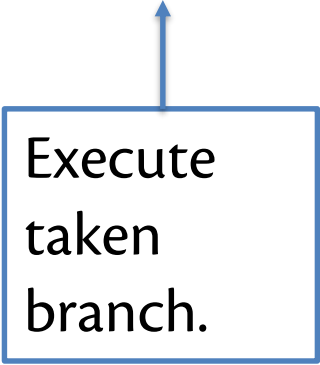


h > 0 is
evaluated
to *false*.

Use on-the-fly static analysis to capture implicit flow

```
x := 0 ;
```

```
if h > 0 then x := 1 else skip
```



Execute
taken
branch.

Use on-the-fly static analysis to capture implicit flow

$x := 0$;

if $h > 0$ then $x := 1$ else skip

On-the-fly static analysis:

$\Gamma(\mathbf{x}) = \Gamma(\mathbf{x}) \sqcup \Gamma(h > 0) = H$

Augment the label of \mathbf{x} with the label context.

Apply on-the-fly static analysis to the untaken branch.

Use on-the-fly static analysis

Goal: \mathbf{x} should be tagged with H at every execution.



```
 $\mathbf{x} := 0 ;$ 
```

```
if  $h > 0$  then  $\mathbf{x} := 1$  else skip
```

$\Gamma(\mathbf{x}) = H$

So, a dynamic mechanism can now deduce labels that correctly capture the flow of information.

But, there is a caveat...

- A dynamic mechanism might leak information
 - when deducing labels during execution, or
 - when deciding to block an execution due to a failed check.
- A static mechanism would not suffer from these leaks.

Leaking through blocking execution

- Consider fixed Γ with $\Gamma(\mathbf{l})=L$ and $\Gamma(\mathbf{h})=H$.
- Consider program:

$\mathbf{l} := 0 ;$

$\text{if } \mathbf{h} > 0 \text{ then } \mathbf{l} := 3 \text{ else } \mathbf{h} := 3 ;$

$\mathbf{l} := 2$

Output

- If $\mathbf{h} > 0$ is *true*, then execution is blocked.
 - No low output.
- If $\mathbf{h} > 0$ is *false*, then execution terminates normally.
 - One low output.
- Thus, $\mathbf{h} > 0$ is leaked to low output.
- How can we solve this problem?

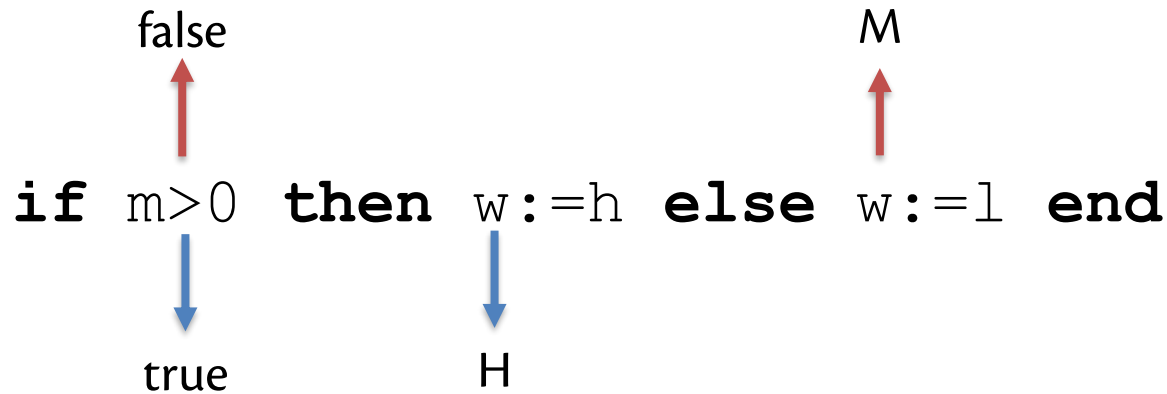
Leaking through blocking execution

- The entire secret might be leaked through blocking.
- Example: consider secret **h** that takes values 1 to 4.

```
l:=1;  
if h=1 then l:=0 else skip;  
l:=2  
if h=2 then l:=0 else skip;  
l:=3  
if h=3 then l:=0 else skip;  
l:=4
```

The final value of **l** equals to **h**!

Leak through flow-sensitive labels



H
L
M
L
L

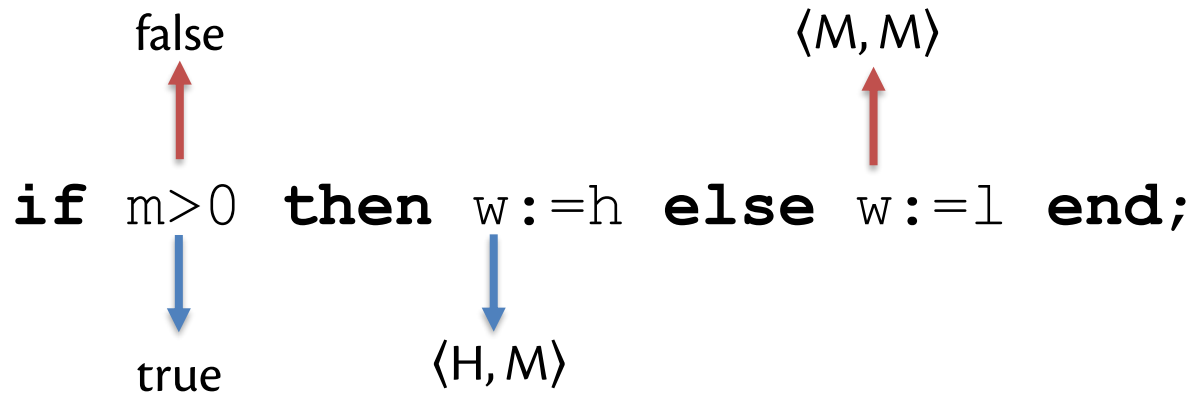
- Flow-sensitive label of `w` always captures the correct sensitivity.
- But `m` leaks to principals reading the flow-sensitive label of `w`.

Strong
Threat
Model

1st solution

- Make flow-sensitive labels independent of guard.
- For our example:
if $m > 0$ **then** $w := h$ **else** $w := 1$ **end**;
tag w always with H at the end of the if-command.
- Prevents leak but introduces conservatism.

2nd solution: Metalabels



But, what is the label of the metalabel of w ?

Leaking through metadata

- Labels, context label, metalabels, etc. are metadata kept by the dynamic mechanism.
- Metadata might encode sensitive information.
- Under a threat model that allows attackers to access metadata, this sensitive information might leak.

Leaking through metadata

- A solution:
 - Add more metadata to protect the existing metadata.
 - Additional metadata can capture information flow with increased precision.
 - Increased permissiveness.
- Because memory is finite, conservatism will be eventually introduced.
 - Some metadata has to conservatively approximate information flow.