
CS 5430

Information flow control (2)

Elisavet Kozyri
Spring 2017

Review: Static type system

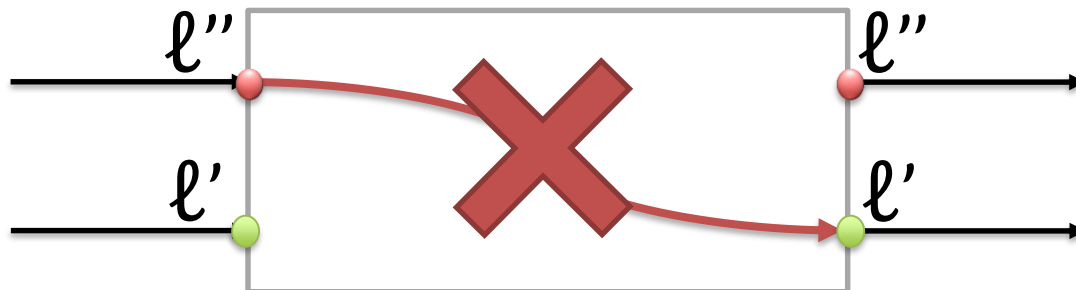
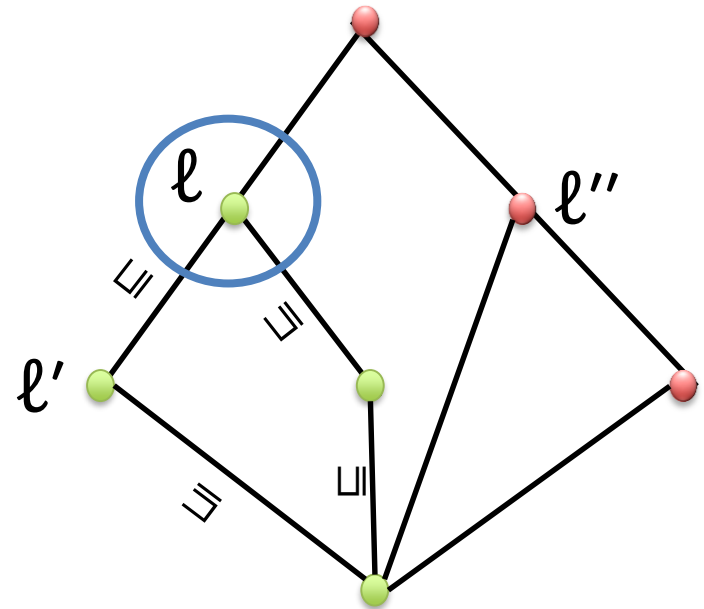
$$\frac{\Gamma \vdash e : \ell \quad \ell \sqcup ctx \sqsubseteq \Gamma(\mathbf{x})}{\Gamma, ctx \vdash \mathbf{x} := e}$$
$$\frac{\Gamma \vdash e : \ell \quad \Gamma, \ell \sqcup ctx \vdash c1 \quad \Gamma, \ell \sqcup ctx \vdash c2}{\Gamma, ctx \vdash \mathbf{if} \ e \ \mathbf{then} \ c1 \ \mathbf{else} \ c2}$$
$$\frac{\Gamma \vdash e : \ell \quad \Gamma, \ell \sqcup ctx \vdash c}{\Gamma, ctx \vdash \mathbf{while} \ e \ \mathbf{do} \ c}$$
$$\frac{\Gamma, ctx \vdash c1 \quad \Gamma, ctx \vdash c2}{\Gamma, ctx \vdash c1 ; c2}$$

Soundness of type system

$\Gamma, ctx \vdash \mathbf{c} \Rightarrow \mathbf{c}$ satisfies NI

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.



Soundness of type system

- Noninterference:
 - $\forall \ell: M_1 =_{\ell} M_2 \Rightarrow \mathbf{c}(M_1) =_{\ell} \mathbf{c}(M_2)$
 - 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 $\ell' \sqsubseteq \ell$.
- $\Gamma, ctx \vdash \mathbf{c} \Rightarrow \mathbf{c}$ 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 does not prevent leaks through covert channels.

Example of covert channel:

 `while s != 0 do { //nothing };
p := 1`

where **s** is a secret variable (i.e., $\Gamma(\mathbf{s})=H$) and **p** is a public variable (i.e., $\Gamma(\mathbf{p})=L$).

- How to represent "do nothing" in our little imperative language?
 - **skip** command
 - i.e., `while s != 0 do skip`
 - Typing rule: $\Gamma, \text{ctx} \vdash \mathbf{skip}$

This type system does not prevent leaks through covert channels.

Example of covert channel:

```
while s != 0 do skip;
```

```
p := 1
```

Output

where **s** is a secret variable and **p** is a public variable.

- If **s != 0** is *true*, then **p := 1** is never executed.
 - No public output!
- If **s != 0** is *false*, then **p** becomes **1**.
 - One public output!
- The termination behavior of the program is used as a *covert channel*, which leaks **s != 0** to public outputs!

This type system does not prevent leaks through covert channels.

Example of covert channel:

```
while s != 0 do skip;  
p:=1
```

where **s** is a secret variable and **p** is a public variable.

- The program leaks over covert channel.
 - It does not satisfy *termination sensitive* noninterference.
- But, the program is type correct.
 - It satisfies (vanilla) noninterference.

A solution

- To prevent covert channels due to infinite loops,
- strengthen the typing rule for while-statement, to allow only **low** guard expression:

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

- Now, type correctness implies termination sensitive NI.
- But, the enforcement mechanism becomes overly conservative.
- Another solution? Research!

Limitations of the type system

MORE



This type system is not complete.

- \mathbf{c} satisfies noninterference $\not\Rightarrow \Gamma, ctx \vdash \mathbf{c}$
 - There is a command \mathbf{c} , such that noninterference is satisfied, but \mathbf{c} is not type correct.
- Example:
 - $\Gamma(\mathbf{x}) = \{\text{Alice}\}, \Gamma(\mathbf{y}) = \{\text{Alice}, \text{Bob}\}$
 - \mathbf{c} is **if $\mathbf{x} > 0$ then $\mathbf{y} := 1$ else $\mathbf{y} := 1$**
 - \mathbf{c} satisfies noninterference, because \mathbf{x} does not leak to \mathbf{y} .
 - \mathbf{c} is not type correct, because $\Gamma(\mathbf{x}) \not\sqsubseteq \Gamma(\mathbf{y})$.

This type system is not complete.

- Another example:
 - $\Gamma(x) = \{\text{Alice}\}$, $\Gamma(y) = \{\text{Alice}, \text{Bob}\}$
 - **c** is **if 1=1 then y:=1 else y:=x**
 - **c** satisfies noninterference, because **x** does not leak to **y**.
 - **c** is not type correct, because $\Gamma(x) \not\subseteq \Gamma(y)$.
- So, this type system is *conservative*.
It has *false positives*:
 - There are programs that satisfy noninterference, but they are not type correct.

This type system has false positives.



Can we build a complete mechanism?

- Is there an enforcement mechanism for information flow control that has no false positives?
 - A mechanism that rejects only programs that do not satisfy noninterference?
- No! [Sabelfeld and Myers, 2003]
 - “The general problem of confidentiality for programs is undecidable.”
 - The halting problem can be reduced to the information flow control problem.
 - Example:
if s>1 then c; p:=2 else skip
 - If we could precisely decide whether this program is secure, we could decide whether **c** terminates!

Can we build a mechanism with fewer false positives?

Switch from static to dynamic mechanisms!

From static to dynamic enforcement mechanisms

- Dynamic mechanisms use run time information to decrease false positives.
- 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 halted 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.

From static to dynamic enforcement mechanisms

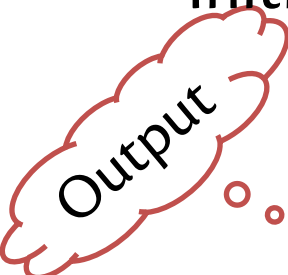
- Under a dynamic enforcement mechanism with fixed Γ ,
- where $\Gamma(\mathbf{x}) = \{\text{Alice}\}$, $\Gamma(\mathbf{y}) = \{\text{Alice}, \text{Bob}\}$,
- command
if 1=1 then y:=1 else y:=x
- would always be executed to completion,
- because dynamic check $\Gamma(\mathbf{1}) \sqcup \Gamma(\mathbf{1=1}) \sqsubseteq \Gamma(\mathbf{y})$ always succeeds,
- and because branch **y:=x** is never taken.
- Remember: the static type system rejects this program before execution, even though the program is secure!

But, there is a caveat...

- A dynamic mechanism may leak information
 - when deducing labels during execution, or
 - when deciding to halt an execution due to a failed check.

Leaking through labels

- *Flow-sensitive* labels: Γ changes during analysis.
- Initially: $\Gamma(\mathbf{x}) = L$, $\Gamma(\mathbf{y}) = L$, $\Gamma(\mathbf{h}) = H$

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

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

$\mathbf{y} := \mathbf{x}$

- At termination, when $\mathbf{h} \neq 0$: $\Gamma(\mathbf{y}) = \Gamma(\mathbf{x}) = L$.
 - One public output.
- At termination, when $\mathbf{h} > 0$: $\Gamma(\mathbf{y}) = \Gamma(\mathbf{x}) = H$.
 - No public output.
- So, $\mathbf{h} > 0$ is leaked to public outputs.
- Problem: Even though \mathbf{h} flows to \mathbf{x} , \mathbf{x} is tagged with L only when $\mathbf{h} \neq 0$.

Leaking through labels

- Purely dynamic mechanisms are usually unsound.
- Purely dynamic mechanism with additional restrictions can become sound:
 - Restriction: Stop execution whenever the guard expression of a conditional command is high.
 - But, the resulting mechanism is more conservative than desired.
- Alternatively...

Use on-the-fly static analysis

- Use on-the-fly static analysis to update the labels of target variables in untaken branch.
- The resulting mechanism is sound and less conservative.

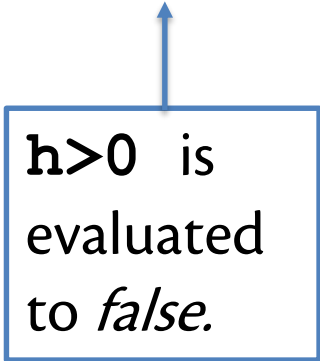
Use on-the-fly static analysis

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

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

```
x := 0 ;
```

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

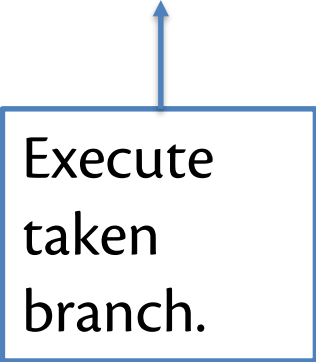


h > 0 is
evaluated
to *false*.

Use on-the-fly static analysis

```
x := 0 ;
```

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



Execute
taken
branch.

Use on-the-fly static analysis

`x := 0 ;`

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

On-the-fly static analysis:

$$\Gamma(\mathbf{x}) = \Gamma(\mathbf{1}) \sqcup \Gamma(\mathbf{h} > 0) = H$$

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.



```
x := 0 ;
```

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

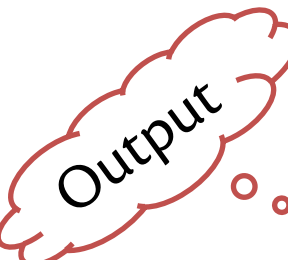
$\Gamma(\mathbf{x}) = H$

But, there is a caveat...

- A dynamic mechanism may leak information
 - when deducing labels during execution, or
 - when deciding to halt an execution (because a check on labels failed).

Leaking through halting execution

- Consider fixed Γ : $\Gamma(\mathbf{p})=L$ and $\Gamma(\mathbf{s})=H$.
- Consider program:

 $\mathbf{p} := 0 ;$
 $\mathbf{if } \mathbf{s} > 0 \mathbf{ then } \mathbf{p} := 1 \mathbf{ else } \mathbf{s} := 1 ;$
 $\mathbf{p} := 2$

- If $\mathbf{s} > 0$ is *true*, then execution is halted.
 - No public output.
- If $\mathbf{s} > 0$ is *false*, then execution terminates normally.
 - One public output.
- Thus, $\mathbf{s} > 0$ is leaked to public outputs.
- How can we solve this problem? Research!

Static versus Dynamic

- Static:
 - Low run time overhead.
 - No new covert channels.
 - More conservative.
- Dynamic
 - Increased run time overhead.
 - Possible new covert channels.
 - Less conservative.
- Ongoing research for both static and dynamic.
 - Different expressiveness of policies, different NI versions, different mechanisms.

Past and current research on static analysis

- [Denning and Denning 1977]
- VSI type system [Volpano, Smith, and Irvine 1996]
- Jif [Myers 1999] Java + Information Flow (originally JFlow)
- FlowCaml [Simonet 2003] OCaml + Information Flow
- Aura, PCML5, Fine, ...

Jif

```
class passwordFile authority(root) {  
  public boolean  
  check (String user, String password)  
  where authority(root) {  
    // Return whether password is correct  
    boolean match = false;  
    try {  
      for (int i = 0; i < names.length; i++) {  
        if (names[i] == user &&  
            passwords[i] == password) {  
          match = true;  
          break;  
        }  
      }  
    }  
    catch (NullPointerException e) {}  
    catch (IndexOutOfBoundsException e) {}  
    return declassify(match, {user; password});  
  }  
  private String [ ] names;  
  private String { root: } [ ] passwords;  
}
```

Jif

```
class passwordFile authority(root) {
  public boolean
  check (String user, String password)
  where authority(root) {
    // Return whether password is correct
    boolean match = false;
    try {
      for (int i = 0; i < names.length; i++) {
        if (names[i] == user &&
            passwords[i] == password) {
          match = true;
          break;
        }
      }
    }
    catch (NullPointerException e) {}
    catch (IndexOutOfBoundsException e) {}
    return declassify(match, {user; password});
  }
  private String [ ] names;
  private String { root: } [ ] passwords;
}
```

Security type:
only `root` may
learn
information in
this field

Jif

```
class passwordFile authority(root) {
  public boolean
  check (String user, String password)
  where authority(root) {
    // Return whether password is correct
    boolean match = false;
    try {
      for (int i = 0; i < names.length; i++) {
        if (names[i] == user &&
            passwords[i] == password) {
          match = true;
          break;
        }
      }
    } catch (NullPointerException e) {}
    catch (IndexOutOfBoundsException e) {}
    return declassify(match, {user; password});
  }
  private String [ ] names;
  private String { root: } [ ] passwords;
}
```

Declassification:

okay to leak
whether
password
matches

Past and current research on dynamic analysis

- RIFLE (ISA) [Vachharajani et al. 2004]
- HiStar (OS) [Zeldovich et al. 2006]
- Trishul (JVM) [Nair et al. 2008]
- TaintDroid (Android) [Enck et al. 2010]
- LIO (Haskell) [Stefan et al. 2011]
- ...

Information flow control: the wheels for security!



Upcoming events

- [Wednesday] A6 due
- [May 18] Final exam

*Suspense is achieved by information control:
What you know. What the reader knows.
What the characters know.
– Tom Clancy*