Secure Programming via Libraries

Soundness of LIO

Alejandro Russo (russo@chalmers.se)

Escuela de Ciencias Informáticas (ECI) 2011
UBA, Buenos Aires, Argentina

Soudness for LIO

[Stefan, Russo, Mitchell, Mazieres 11]

- Formalizes the non-interference guarantee provided by LIO
- For the proof, we consider a core and simple and functional language
  - Why not full Haskell?
  - λ-calculus extended with boolean values, pairs, recursion, monadic operations, references
- *We formally prove that the concept of monads works to guarantee non-interference*

Proof Technique

- Similar technique as the one used by Li and Zdancewic [Li, Zdancewic 10]
- Programs are expressions
- Main idea is simple:
  - If a program, that **involves secret and public information**, computes a public result, then the same public result can be obtained by a program that consists on the original one where **the secret data has been erased**!

Proof Technique

- More technically, we build a simulation between

Program with secret (e.g. `Labeled H Int`) and public data

Reduces one step

$$e_1 \longrightarrow e_2$$

Program with secret (e.g. `Labeled H Int`) and public data

and

Program where secrets where erased

Reduces one step

$$\varepsilon_L(e_1) \longrightarrow_L \varepsilon_L(e_2)$$

Program where secrets where erased

The Language

The language

- The language and types

$$
\begin{array}{rrl}
\text{Label:} & & l \\
\text{Address:} & & a \\
\text{Term:} & v ::= & \texttt{true} \mid \texttt{false} \mid \texttt{()} \mid l \mid a \mid x \mid \lambda x.e \mid (e, e) \\
& & \mid \texttt{fix}\ e \mid \texttt{Lb}\ v\ e \mid (e)^{\texttt{LIO}} \mid \bullet \\
\text{Expression:} & e ::= & v \mid e\ e \mid \pi_i\ e \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
& & \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \texttt{return}\ e \mid e\ \texttt{>>=}\ e \mid \ldots \\
\text{Type:} & \tau ::= & \texttt{Bool} \mid \texttt{()} \mid \tau \to \tau \mid (\tau, \tau) \\
& & \mid \ell \mid \texttt{Labeled}\ \ell\ \tau \mid \texttt{LIO}\ \ell\ \tau \mid \texttt{Ref}\ \ell\ \tau \\
\text{Store:} & & \phi : \text{Address} \to \texttt{Labeled}\ \ell\ \tau
\end{array}
$$

The language

- The language and types

Special syntax node: *internal representation LIO computations*

Special syntax node: *it represents term erasure*

Special syntax node: *internal representation of Labeled values*

$$
\begin{array}{lrl}
\text{Address:} & & a \\
\text{Term:} & v ::= & \texttt{true} \mid \texttt{false} \mid () \mid l \mid a \mid x \mid \lambda x.e \mid (e, e) \\
& & \mid \texttt{fix}\ e \mid \texttt{Lb}\ v\ e \mid (e)^{\texttt{LIO}} \mid \bullet \\
\text{Expression:} & e ::= & v \mid e\ e \mid \pi_i\ e \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \\
& & \mid \texttt{let}\ x = e\ \texttt{in}\ e \mid \texttt{return}\ e \mid e \mathbin{\texttt{>>=}} e \mid \ldots \\
\text{Type:} & \tau ::= & \texttt{Bool} \mid () \mid \tau \to \tau \mid (\tau, \tau) \\
& & \mid \ell \mid \texttt{Labeled}\ \ell\ \tau \mid \texttt{LIO}\ \ell\ \tau \mid \texttt{Ref}\ \ell\ \tau \\
\text{Store:} & & \phi : \text{Address} \to \texttt{Labeled}\ \ell\ \tau
\end{array}
$$

return and $>>=$

- Every monad has two operations: return and bind

$$\texttt{return} :: a \rightarrow M a$$
$$\texttt{>>=} :: M a \rightarrow (a \rightarrow M b) \rightarrow M b$$

- So far, we wrote programs using **do**

$$e_1 \texttt{ >>= } \lambda x . e_2 \quad \Rightarrow$$

```
do x ← e1
   e2
```

The Semantics

Operational Semantics

- It describes how a valid program is interpreted as a sequence of computational steps [Winskel]
- We describe the steps via evaluation contexts
- **Evaluation contexts**
  - An evaluation contexts E is just a term with a "hole"
  - $E[e]$ is the substitution of e into the hole
  - Intuitively, if a term M is being evaluated where $M = E[e]$
    - E is the context
    - e is the part of the term being evaluated

Evaluation Example

Expression to evaluate

$$e ::= \texttt{true} \mid \texttt{false} \mid (e, e) \mid \pi_i\ e \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{el}$$

$$E ::= [\cdot] \mid \pi_i\ E \mid \texttt{if}\ E\ \texttt{then}\ e\ \texttt{else}\ e \mid \ldots$$

$$M = \texttt{if}\ \pi_1\ (\texttt{true}, \texttt{false})\ \texttt{then}\ \texttt{true}\ \texttt{else}\ (\texttt{false}, \texttt{true})$$

$$E_1 = \texttt{if}\ [\cdot]\ \texttt{then}\ \texttt{true}\ \texttt{else}\ (\texttt{false}, \texttt{true})$$

$$M = E_1[\pi_1\ (\texttt{true}, \texttt{false})]$$

Expressed in terms of evaluation contexts

$$E_1[\pi_1\ (\texttt{true}, \texttt{false})] \longrightarrow E_1[\texttt{true}]$$

Reduction step

$$E_1[\texttt{true}] = \texttt{if}\ \texttt{true}\ \texttt{then}\ \texttt{true}\ \texttt{else}\ (\texttt{false}, \texttt{true})$$

$$E_1[\texttt{true}] = [\cdot][\texttt{if}\ \texttt{true}\ \texttt{then}\ \texttt{true}\ \texttt{else}\ (\texttt{false}, \texttt{true})]$$

$$[\cdot][\texttt{if}\ \texttt{true}\ \texttt{then}\ \texttt{true}\ \texttt{else}\ (\texttt{false}, \texttt{true})] \longrightarrow [\cdot][\texttt{true}] = \texttt{true}$$

$$M \longrightarrow^* \texttt{true}$$

Reduction rules

$$E[\pi_i\ (e_1, e_2)] \longrightarrow E[e_i]$$

$$E[\texttt{if}\ \texttt{true}\ \texttt{then}\ e_1\ \texttt{else}\ e_2] \longrightarrow E[e_1]$$

$$E[\texttt{if}\ \texttt{false}\ \texttt{then}\ e_1\ \texttt{else}\ e_2] \longrightarrow E[e_2]$$

Operational Semantics for LIO

$$v ::= \ \ldots \mid \texttt{Lb}\ v\ e \mid \ldots$$

$$e ::= \ \ldots \mid \texttt{label}\ e\ e \mid \texttt{unlabel}\ e \mid \texttt{toLabeled}\ e\ e$$
$$\mid \texttt{newRef}\ e\ e \mid \ldots$$

$$E ::= \ [\cdot] \mid \ldots$$

- LIO computations have state
  - Current label, clearance, and an store for references

State of the LIO computation

$$\langle \Sigma, E[e] \rangle \longrightarrow \langle \Sigma', E[e'] \rangle$$

Reduction step

Current label

Current clearance

Store

$\Sigma.\texttt{lbl}$ $\Sigma.\texttt{clr}$ $\Sigma.\phi$

Operational Semantics for LIO

- The security checks are done in the semantics
  - Dynamic approach

It respects the current label and clearance

(LAB)

$$\frac{\Sigma.\mathtt{lbl} \sqsubseteq l \sqsubseteq \Sigma.\mathtt{clr}}{\langle \Sigma, E[\mathtt{label}\ l\ e] \rangle \longrightarrow \langle \Sigma, E[\mathtt{return}\ (\mathtt{Lb}\ l\ e)] \rangle}$$

If the security checks are not fulfilled, the execution gets “stuck”.
In practice, it could be an uncaught exception, etc.

It evaluates to the internal representation

Operational Semantics for LIO

It is the join of the current label and the label that protects e

Clearance is respected

It sets a new current label

(UNLAB)

$$\frac{l' = \Sigma.\mathtt{lbl} \sqcup l \qquad l' \sqsubseteq \Sigma.\mathtt{clr} \qquad \Sigma' = \Sigma[\mathtt{lbl} \mapsto l']}{\langle \Sigma, E[\mathtt{unlabel}\ (\mathtt{Lb}\ l\ e)] \rangle \longrightarrow \langle \Sigma', E[\mathtt{return}\ e] \rangle}$$

A `Labeled` value which contents is e

It extracts the value e and returns it

Operational Semantics for LIO

The current label after executing e should be below l

The label of the result is among the current label and clearance

It executes the LIO computation e

(OLAB)

$$\frac{\Sigma.\mathtt{lbl} \sqsubseteq l \sqsubseteq \Sigma.\mathtt{clr} \qquad \langle \Sigma, e \rangle \longrightarrow^{*} \langle \Sigma', (v)^{\mathtt{LIO}} \rangle \qquad \Sigma'.\mathtt{lbl} \sqsubseteq l \qquad \Sigma'' = \Sigma'[\mathtt{lbl} \mapsto \Sigma.\mathtt{lbl}, \mathtt{clr} \mapsto \Sigma.\mathtt{clr}]}{\langle \Sigma, E[\mathtt{toLabeled}\ l\ e] \rangle \longrightarrow \langle \Sigma'', E[\mathtt{label}\ l\ v] \rangle}$$

The label of the result of computing e

Observe that this state has (only) the same current label and clearance values as when starting executing e

Operational Semantics for LIO

The allocated memory location is "new"

The store in the state gets modified

(NREF)

$$\frac{a \textit{ fresh} \qquad \Sigma.\mathtt{lbl} \sqsubseteq l \sqsubseteq \Sigma.\mathtt{clr} \qquad \Sigma' = \Sigma.\phi[a \mapsto \mathtt{Lb}\ l\ e]}{\langle \Sigma, E[\mathtt{newRef}\ l\ e] \rangle \longrightarrow \langle \Sigma', E[\mathtt{return}\ a] \rangle}$$

It returns a memory location

Operational Semantics for LIO

- You have seen a few rules
- Check the paper for the rest of them
  [Stefan, Russo, Mitchell, Mazieres 11]
- You should be able to understand them after the lecture

The Types

Typing

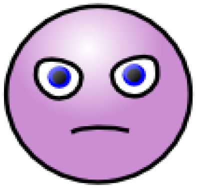
- It is not very interesting for our library
  - It is a dynamic approach, not static one
- Typing judgment

Expression to type

Assumptions about the type of free variables in e

$$\Gamma \vdash e : \tau$$

Type



$$\Gamma = [x : \texttt{Bool}] \quad \Gamma \vdash x + x : ?$$

$$\Gamma \vdash x + x : ?$$

$$\Gamma = [x : Int] \quad \Gamma \vdash x + x : Int$$

Typing rules

- They indicate how to perform type-checking
  - Rules are usually syntax-directed rules
- An expression type-checks if we can construct a *type derivation* (application of the typing rules)

Type system (very simple)

$$\Gamma \vdash 1 : \mathtt{Int}$$

$$\Gamma \vdash \mathtt{true} : \mathtt{Bool}$$

$$\frac{\Gamma \vdash e : \tau \qquad \Gamma \vdash e' : \tau'}{\Gamma \vdash (e, e') : (\tau, \tau')}$$

Here you have the type derivation!

$$\frac{\dfrac{\Gamma \vdash \mathtt{true} : \mathtt{Bool} \quad \Gamma \vdash 1 : \mathtt{Int}}{\Gamma \vdash (\mathtt{true}, 1) : (\mathtt{Bool}, \mathtt{Int})} \qquad \Gamma \vdash \mathtt{true} : \mathtt{Bool}}{\Gamma \vdash ((\mathtt{true}, 1), \mathtt{true}) : ((\mathtt{Bool}, \mathtt{Int}), \mathtt{Bool})}$$

What is the type?

Interesting typing rules

Special syntax node: *internal representation of Labeled values*

Special syntax node: *internal representation LIO computations*

Special syntax node: *it represents term erasure*

Term: $v ::= \ \dots \mid \texttt{Lb}\ v\ e \mid (e)^{\texttt{LIO}} \mid \bullet$

Store: $\phi : \text{Address} \to \texttt{Labeled}\ \ell\ \tau$

$$\frac{\Gamma \vdash e_1 : \ell \qquad \Gamma \vdash e_2 : \tau}{\Gamma \vdash \texttt{Lb}\ e_1\ e_2 : \texttt{Labeled}\ \ell\ \tau} \qquad \frac{\Gamma \vdash e : \tau}{\Gamma \vdash (e)^{\texttt{LIO}} : \texttt{LIO}\ \ell\ \tau}$$

$$\Gamma \vdash \bullet : \tau$$

- The rest of the typing rules are just like the ones implemented in Haskell

So far

- We have seen
  - The language
  - Semantics
  - Types
- What is coming now?
  - Combine all of them (and some other techniques) in order to prove non-interference in programs written using LIO

Soundness

Proof Technique

- More technically, we build a simulation between

Program with secret (e.g. `Labeled H Int`) and public data

Reduces one step

$$e_1 \longrightarrow e_2$$

Program with secret (e.g. `Labeled H Int`) and public data

and

Program where secrets where erased

Reduces one step

$$\varepsilon_L(e_1) \longrightarrow_L \varepsilon_L(e_2)$$

Program where secrets where erased

The Erasure Function

- Function ε_L
  - It is responsible for performing *term erasure*
  - It is often applied homomorphically

$$\varepsilon_L(\texttt{if}\ e\ \texttt{then}\ e_1\ \texttt{else}\ e_2) =$$
$$\texttt{if}\ \varepsilon_L(e)\ \texttt{then}\ \varepsilon_L(e_1)\ \texttt{else}\ \varepsilon_L(e_2)$$

- Intuitively, the function removes values and expressions that are not below L
- L is the attacker level

The Erasure Function

$$\varepsilon_L(\bullet) = \bullet$$

Idempotent

$$\varepsilon_L((e)^{\texttt{LIO}}) = (\varepsilon_L(e))^{\texttt{LIO}}$$

$$\varepsilon_L(\texttt{Lb}\; l\; e) = \begin{cases} \texttt{Lb}\; l\; \bullet & l \not\sqsubseteq L \\ \texttt{Lb}\; l\; \varepsilon_L(e) & \text{otherwise} \end{cases}$$

It removes labeled values where the label Is not below L

$$\frac{\varepsilon_L(\Sigma.\phi) = \{(x, \varepsilon_L(\Sigma.\phi(x)) : x \in \mathrm{dom}(\Sigma.\phi)\}}{\varepsilon_L(\Sigma) = \Sigma[\phi \mapsto \varepsilon_L(\Sigma.\phi)]}$$

It propagates the application of the erasure function to the labeled values stored by references

$$\varepsilon_L(\langle \Sigma, e \rangle) = \begin{cases} \langle \varepsilon_L(\Sigma), \bullet \rangle & \Sigma.\texttt{lbl} \not\sqsubseteq L \\ \langle \varepsilon_L(\Sigma), \varepsilon_L(e) \rangle & \text{otherwise} \end{cases}$$

Erasure in configurations (technical reasons)

Proof Technique

- More technically, we build a simulation between

Program with secret (e.g. `Labeled H Int`) and public data

Reduces one step

$$e_1 \longrightarrow e_2$$

Program with secret (e.g. `Labeled H Int`) and public data

and

Program where secrets where erased

Reduces one step

$$\varepsilon_L(e_1) \longrightarrow_L \varepsilon_L(e_2)$$

Program where secrets where erased

A new evaluation relationship

$$\frac{\langle \Sigma, e \rangle \longrightarrow \langle \Sigma', e' \rangle}{\langle \Sigma, e \rangle \longrightarrow_L \varepsilon_L(\langle \Sigma', e' \rangle)}$$

- Expressions under this evaluation relationship are evaluated as before
- It guarantees that confidential data (above L) is erased as soon as it is created

Simulation

- This is the main idea behind the proof

$$
\begin{tikzcd}
\langle \Sigma, e \rangle \arrow[r] \arrow[d, "\varepsilon_L"] & {}^{*} \; \langle \Sigma', e' \rangle \arrow[d, "\varepsilon_L"] \\
\varepsilon_L(\langle \Sigma, e \rangle) \arrow[r] & {}^{*}_{L} \; \varepsilon_L(\langle \Sigma', e' \rangle)
\end{tikzcd}
$$

Preliminaries

- In order to prove the simulation, it is necessary to show several auxiliary results
  - You can read it from the paper
- The proof consists on establishing the simulation in two phases
  - For expressions that do not execute any `toLabeled`
  - For expressions that execute n-`toLabeled`
- Why is that?
  - The semantics for `toLabeled` uses big-step semantics

Establishing the simulation

Lemma 1 (Single-step simulation without `toLabeled`).
If

- $\Gamma \vdash e : \tau$, *and*

Subject reductoin

- $\langle \Sigma, e \rangle \longrightarrow \langle \Sigma', e' \rangle$

where `toLabeled` *is not executed, then*

i) $\Gamma \vdash e' : \tau$, *and*

Subject reductoin

ii) $\varepsilon_L(\langle \Sigma, e \rangle) \longrightarrow_L \varepsilon_L(\langle \Sigma', e' \rangle)$.

Establishing the simulation

- The proof going on case analysis on the expression being evaluated
  - Recall that evaluation is performed using evaluation contexts

Establishing the simulation

Case:

$$\frac{\Sigma.\texttt{lbl} \sqsubseteq l \sqsubseteq \Sigma.\texttt{clr}}{\langle \Sigma, E[\texttt{label}\ l\ e] \rangle \longrightarrow \langle \Sigma, E[\texttt{return}\ (\texttt{Lb}\ l\ e)] \rangle}:$$

- $l \sqsubseteq L$:

It applies the definition in a left-to-right manner

It just applies the definition

Idempotent erasure function

$$\begin{aligned}
& \varepsilon_L(\langle \Sigma, E[\texttt{label}\ l\ e] \rangle) \\
&= \langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\texttt{label}\ l\ \varepsilon_L(e)] \rangle \\
&\longrightarrow_L \varepsilon_L(\langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\texttt{return}\ (\texttt{Lb}\ l\ \varepsilon_L(e))] \rangle) \\
&= \langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\texttt{return}\ (\texttt{Lb}\ l\ \varepsilon_L(e))] \rangle \\
&= \langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\varepsilon_L(\texttt{return}\ (\texttt{Lb}\ l\ e))] \rangle \\
&= \varepsilon_L(\langle \Sigma, E[\texttt{return}\ (\texttt{Lb}\ l\ e)] \rangle)
\end{aligned}$$

It applies the definition in a right-to-left manner

Establishing the simulation

Case:

$$\frac{\Sigma.\texttt{lbl} \sqsubseteq l \sqsubseteq \Sigma.\texttt{clr}}{\langle \Sigma, E[\texttt{label}\ l\ e] \rangle \longrightarrow \langle \Sigma, E[\texttt{return}\ (\texttt{Lb}\ l\ e)] \rangle}:$$

- $l \not\sqsubseteq L$:

It applies the definition in a left-to-right manner

It just applies the definition

Idempotent erasure function

$$\begin{aligned}
&\varepsilon_L(\langle \Sigma, E[\texttt{label}\ l\ e] \rangle) \\
&= \langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\texttt{label}\ l\ \varepsilon_L(e)] \rangle \\
\longrightarrow_L\ &\varepsilon_L(\langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\texttt{return}\ (\texttt{Lb}\ l\ \varepsilon_L(e))] \rangle) \\
&= \langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\texttt{return}\ (\texttt{Lb}\ l\ \bullet)] \rangle \\
&= \langle \varepsilon_L(\Sigma), \varepsilon_L(E)[\varepsilon_L(\texttt{return}\ (\texttt{Lb}\ l\ e))] \rangle \\
&= \varepsilon_L(\langle \Sigma, E[\texttt{return}\ (\texttt{Lb}\ l\ e)] \rangle)
\end{aligned}$$

It applies the definition in a right-to-left manner

Establishing the simulation

Lemma 2 (Simulation for expressions not executing `toLabeled`). *If*

- $\Gamma \vdash e : \tau$, *and*
- $\langle \Sigma, e \rangle \longrightarrow^{*} \langle \Sigma', e' \rangle$

where `toLabeled` *is not executed, then*

i) $\Gamma \vdash e' : \tau$, *and*

ii) $\varepsilon_L(\langle \Sigma, e \rangle) \longrightarrow_L^{*} \varepsilon_L(\langle \Sigma', e' \rangle)$.

- The proof is on induction on $\longrightarrow^{*}$
- The base case is Lemma 1

Establishing the simulation

Lemma 3 (Simulation). *If* $\Gamma \vdash e : \tau$ *and* $\langle \Sigma, e \rangle \longrightarrow^{*} \langle \Sigma', e' \rangle$ *then* $\Gamma \vdash e' : \tau$ *and* $\varepsilon_L(\langle \Sigma, e \rangle) \longrightarrow_L^{*} \varepsilon_L(\langle \Sigma', e' \rangle)$.

- The proof is on induction on the number of `toLabeled` being executed
- Base case is Lemma 2
- For the inductive case, we rewrite the big-step semantics into

≤ k `toLabeled`

no `toLabeled`

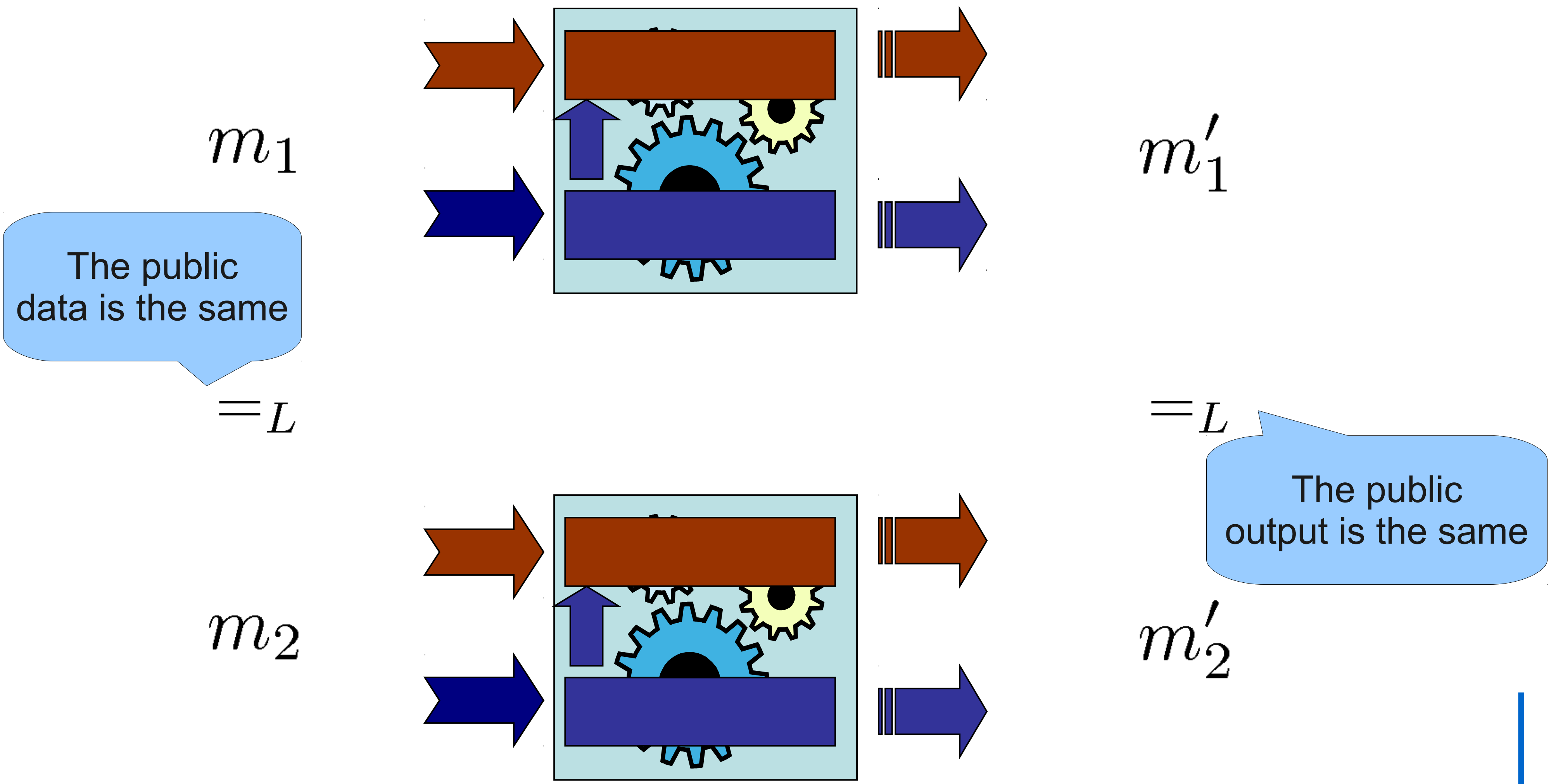
≤ k `toLabeled`

$$\overbrace{\langle \Sigma, e \rangle \longrightarrow^{*} \langle \Sigma_0, E[}^{\text{first big-step}}\overbrace{\texttt{toLabeled}\; l\; e_0]\rangle \longrightarrow^{*} \langle \Sigma', e' \rangle}^{\text{second big-step}}$$

Non-interference

- Having the simulation established
- We proceed with a formulation of the theorem that proves non-interference
- The formulation is “standard”
- It requires a notion of low-equivalence
- It captures the observational power of the attacker
- If we run the program twice but with the same public input, the same public output must be observed

Low-equivalence



Low-equivalence

- We considered labeled values as the input and output of programs
- Intuitively, two expressions are low-equivalent if the are equal, modulo labeled values whose labels are above L

$$\frac{e \approx_L e' \qquad l \sqsubseteq L}{\mathtt{Lb}\ l\ e \approx_L \mathtt{Lb}\ l\ e'} \qquad\qquad \frac{l \not\sqsubseteq L}{\mathtt{Lb}\ l\ e \approx_L \mathtt{Lb}\ l\ e'}$$

If the label is not below L, then the content of labeled values it is not important

$\mathtt{if\ true\ then\ (Lb}\ H\ \mathtt{false)\ else\ false} \approx_L$
$\mathtt{if\ true\ then\ (Lb}\ H\ \mathtt{true)\ else\ false}$

Low-equivalence

- We define low-equivalence between stores as well
- Intuitively, two stores are low-equivalent if the stored labeled values below L are the same

Both stores contains the same *public* labeled values

The *public* labeled values are low-equivalent

$$\frac{\mathrm{dom}_L\,(\Sigma.\phi) = \mathrm{dom}_L\,(\Sigma.\phi') \qquad \forall a \in \mathrm{dom}_L(\Sigma.\phi) \cdot \Sigma.\phi(a) \approx_L \Sigma'.\phi(a)}{\Sigma.\phi \approx_L \Sigma'.\phi}$$

Low-equivalence

- We now define low-equivalence for configurations
  - It essentially means to have low-equivalence in the store and the expression to be evaluated when the current label is below L

$$\frac{\Sigma.\phi \approx_L \Sigma'.\phi \qquad \Sigma.\texttt{lbl} = \Sigma'.\texttt{lbl} \qquad \Sigma.\texttt{clr} = \Sigma'.\texttt{clr} \qquad \Sigma.\texttt{lbl} \sqsubseteq L \qquad e \approx_L e'}{\langle \Sigma, e \rangle \approx_L \langle \Sigma', e' \rangle}$$

$$\frac{\Sigma.\phi \approx_L \Sigma'.\phi \qquad \Sigma.\texttt{lbl} \not\sqsubseteq L \qquad \Sigma'.\texttt{lbl} \not\sqsubseteq L}{\langle \Sigma, e \rangle \approx_L \langle \Sigma', e' \rangle}$$

Non-interference

Theorem 1 (Non-interference). *Given a computation e (with no $\bullet$, $(\ )^{\mathtt{LIO}}$, or $\mathtt{Lb}$) where $\Gamma \vdash e : \mathtt{Labeled}\,\ell\,\tau \rightarrow \mathtt{LIO}\,\ell\,(\mathtt{Labeled}\,\ell\,\tau')$, initial environments Σ_1 and Σ_2 where $\Sigma_1.\phi = \Sigma_2.\phi = \emptyset$, security label l, an attacker at level L such that $l \sqsubseteq L$, then*

$$\begin{aligned}
&\forall e_1 e_2 . (\Gamma \vdash e_i : \mathtt{Labeled}\,\ell\,\tau)_{i=1,2} \\
&\qquad \wedge (e_i = \mathtt{Lb}\,l\ e_i')_{i=1,2} \wedge \langle \Sigma_1, e\ e_1 \rangle \approx_L \langle \Sigma_2, e\ e_2 \rangle \\
&\qquad \wedge \langle \Sigma_1, e\ e_1 \rangle \longrightarrow^* \langle \Sigma_1', (\mathtt{Lb}\,l_1\ e_1'')^{\mathtt{LIO}} \rangle \\
&\qquad \wedge \langle \Sigma_2, e\ e_2 \rangle \longrightarrow^* \langle \Sigma_2', (\mathtt{Lb}\,l_2\ e_2'')^{\mathtt{LIO}} \rangle \\
&\qquad \Rightarrow \langle \Sigma_1', \mathtt{Lb}\,l_1\ e_1'' \rangle \approx_L \langle \Sigma_2', \mathtt{Lb}\,l_2\ e_2'' \rangle
\end{aligned}$$

Non-interference (specialized)

Theorem 1 (Non-interference). *Given a computation e (with no $\bullet$, $(\,)^{\texttt{LIO}}$, or $\texttt{Lb}$) where $\Gamma \vdash e : \texttt{Labeled}\,\ell\,\tau \rightarrow \texttt{LIO}\,\ell\,(\texttt{Labeled}\,\ell\,\tau')$, initial environments Σ_1 and Σ_2 where $\Sigma_1.\phi = \Sigma_2.\phi = \emptyset$, an attacker at level L, then*

$$
\begin{aligned}
&\forall e_1 e_2.(\Gamma \vdash e_i : \texttt{Labeled}\,\ell\,\tau)_{i=1,2} \\
&\qquad \wedge (e_i = \texttt{Lb}\,H\;e_i')_{i=1,2} \wedge \langle \Sigma_1, e\;e_1\rangle \approx_L \langle \Sigma_2, e\;e_2\rangle \\
&\qquad \wedge \langle \Sigma_1, e\;e_1\rangle \longrightarrow^* \langle \Sigma_1', (\texttt{Lb}\,l_1\;e_1'')^{\texttt{LIO}}\rangle \\
&\qquad \wedge \langle \Sigma_2, e\;e_2\rangle \longrightarrow^* \langle \Sigma_2', (\texttt{Lb}\,l_2\;e_2'')^{\texttt{LIO}}\rangle \\
&\qquad \Rightarrow \langle \Sigma_1', \texttt{Lb}\,l_1\;e_1''\rangle \approx_L \langle \Sigma_2', \texttt{Lb}\,l_2\;e_2''\rangle
\end{aligned}
$$

It should have use $(e_i = \texttt{Lb}\;L\;(\texttt{Lb}\;H\;e_i'))_{i=1,2}$
but for simplicity I did not

Proof Sketch

- We will use our simulation
- We asumme (you can prove it) that

$$\varepsilon_L(e) = \varepsilon_L(e') \Rightarrow e \approx_L e'$$

Proof Sketch II

$$(e_i = \mathtt{Lb}\ H\ e_i')_{i=1,2} \wedge \langle \Sigma_1, e\ e_1 \rangle \approx_L \langle \Sigma_2, e\ e_2 \rangle$$
$$\wedge \langle \Sigma_1, e\ (\mathtt{Lb}\ H\ e_1') \rangle \longrightarrow^* \langle \Sigma_1', (\mathtt{Lb}\ l_1\ e_1'')^{\mathtt{LIO}} \rangle$$
$$\wedge \langle \Sigma_2, e\ (\mathtt{Lb}\ H\ e_2') \rangle \longrightarrow^* \langle \Sigma_2', (\mathtt{Lb}\ l_2\ e_2'')^{\mathtt{LIO}} \rangle$$

- By our simulation, we know that

By the simulation

$$\varepsilon_L(\langle \Sigma_1, e\ (\mathtt{Lb}\ H\ e_1') \rangle) \longrightarrow_L^* \varepsilon_L(\langle \Sigma_1', (\mathtt{Lb}\ l_1\ e_1'')^{\mathtt{LIO}} \rangle)$$
$$\varepsilon_L(\langle \Sigma_2, e\ (\mathtt{Lb}\ H\ e_2') \rangle) \longrightarrow_L^* \varepsilon_L(\langle \Sigma_2', (\mathtt{Lb}\ l_2\ e_2'')^{\mathtt{LIO}} \rangle)$$

Proof Sketch III

$$\varepsilon_L(\langle \Sigma_1, e\ (\mathtt{Lb}\ H\ e_1')\rangle) \longrightarrow_L^* \varepsilon_L(\langle \Sigma_1', (\mathtt{Lb}\ l_1\ e_1'')^{\mathtt{LIO}}\rangle)$$
$$\varepsilon_L(\langle \Sigma_2, e\ (\mathtt{Lb}\ H\ e_2')\rangle) \longrightarrow_L^* \varepsilon_L(\langle \Sigma_2', (\mathtt{Lb}\ l_2\ e_2'')^{\mathtt{LIO}}\rangle)$$

Erase function goes inside the configuration

- We expand it

$$\langle \varepsilon_L(\Sigma_1), \varepsilon_L(e\ (\mathtt{Lb}\ H\ e_1'))\rangle \longrightarrow^* \varepsilon_L(\langle \Sigma_1', (\mathtt{Lb}\ l_1\ e_1'')^{\mathtt{LIO}}\rangle)$$
$$\langle \varepsilon_L(\Sigma_2), \varepsilon_L(e\ (\mathtt{Lb}\ H\ e_2'))\rangle \longrightarrow^* \varepsilon_L(\langle \Sigma_2', (\mathtt{Lb}\ l_2\ e_2'')^{\mathtt{LIO}}\rangle)$$

- A little bit more

$$\langle \varepsilon_L(\Sigma_1), \varepsilon_L(e)\ (\mathtt{Lb}\ H\ \bullet)\rangle \longrightarrow^* \varepsilon_L(\langle \Sigma_1', (\mathtt{Lb}\ l_1\ e_1'')^{\mathtt{LIO}}\rangle)$$
$$\langle \varepsilon_L(\Sigma_2), \varepsilon_L(e)\ (\mathtt{Lb}\ H\ \bullet)\rangle \longrightarrow^* \varepsilon_L(\langle \Sigma_2', (\mathtt{Lb}\ l_2\ e_2'')^{\mathtt{LIO}}\rangle)$$

Proof Sketch IV

These are the same configurations

$$\langle \varepsilon_L(\Sigma_1), \varepsilon_L(e)\ (\texttt{Lb}\ H\ \bullet)\rangle \longrightarrow_L^* \varepsilon_L(\langle \Sigma_1', (\texttt{Lb}\ l_1\ e_1'')^{\texttt{LIO}}\rangle)$$
$$\langle \varepsilon_L(\Sigma_2), \varepsilon_L(e)\ (\texttt{Lb}\ H\ \bullet)\rangle \longrightarrow_L^* \varepsilon_L(\langle \Sigma_2', (\texttt{Lb}\ l_2\ e_2'')^{\texttt{LIO}}\rangle)$$

By equality and definition of erasure function

- We know that $\longrightarrow_L^*$ is deterministic
- Then,

$$\varepsilon_L(\langle \Sigma_1', (\texttt{Lb}\ l_1\ e_1'')^{\texttt{LIO}}\rangle) = \varepsilon_L(\langle \Sigma_2', (\texttt{Lb}\ l_2\ e_2'')^{\texttt{LIO}}\rangle)$$

Remember what we assume in the begining

- Which means,

By definition of erasure function

$$\varepsilon_L((\texttt{Lb}\ l_1\ e_1'')^{\texttt{LIO}}) = \varepsilon_L((\texttt{Lb}\ l_2\ e_2'')^{\texttt{LIO}})$$

$$\varepsilon_L(\texttt{Lb}\ l_1\ e_1'') = \varepsilon_L(\texttt{Lb}\ l_2\ e_2'') \qquad \Rightarrow \texttt{Lb}\ l_1\ e_1'' \approx_L \texttt{Lb}\ l_2\ e_2''$$

Proof Sketch V

- Then,

$$\varepsilon_L(\langle \Sigma_1', (\mathtt{Lb}\ l_1\ e_1'')^{\mathtt{LIO}} \rangle) = \varepsilon_L(\langle \Sigma_2', (\mathtt{Lb}\ l_2\ e_2'')^{\mathtt{LIO}} \rangle)$$

- Which means,

By equality and definition of erasure function

$$\varepsilon_L(\Sigma_1'.\phi) = \varepsilon_L(\Sigma_2'.\phi) \quad \Rightarrow \mathrm{dom}_L\,(\Sigma_1'.\phi) = \mathrm{dom}_L\,(\Sigma_2'.\phi)$$

- For any "public" labeled value in the store, we have

$$\varepsilon_L(\Sigma_1'.\phi(x)) = \varepsilon_L(\Sigma_2'.\phi(x)),\ \text{for any } x \in \mathrm{dom}_L\,(\Sigma_1'.\phi)$$

By definition of erasure function and equality

$$\Rightarrow \Sigma_1'.\phi(x) \approx_L \Sigma_2'.\phi(x),\ \text{for any } x \in \mathrm{dom}_L\,(\Sigma_1'.\phi)$$

What we assume in the beginning

$$\Rightarrow \Sigma_1'.\phi \approx_L \Sigma_2'.\phi$$

By definition of low-equivalence for stores

Proof Sketch VI

- Now, we have that

$$\Sigma_1'.\phi \approx_L \Sigma_2'.\phi \qquad \texttt{Lb}\ l_1\ e_1'' \approx_L \texttt{Lb}\ l_2\ e_2''$$

- We still need to prove

$$\langle \Sigma_1', \texttt{Lb}\ l_1\ e_1'' \rangle \approx_L \langle \Sigma_2', \texttt{Lb}\ l_2\ e_2'' \rangle$$

- From the simulation, we had

$$\varepsilon_L(\langle \Sigma_1', (\texttt{Lb}\ l_1\ e_1'')^{\texttt{LIO}} \rangle) = \varepsilon_L(\langle \Sigma_2', (\texttt{Lb}\ l_2\ e_2'')^{\texttt{LIO}} \rangle)$$

- Which implies that

$$\Sigma_1'.\texttt{lbl} = \Sigma_2'.\texttt{lbl} \wedge \Sigma_1'.\texttt{clr} = \Sigma_2'.\texttt{clr}$$

Proof Sketch VII

- So, having

$$\Sigma'_1.\phi \approx_L \Sigma'_2.\phi \qquad \texttt{Lb}\ l_1\ e''_1 \approx_L \texttt{Lb}\ l_2\ e''_2$$

$$\Sigma'_1.\texttt{lbl} = \Sigma'_2.\texttt{lbl} \qquad \Sigma'_1.\texttt{clr} = \Sigma'_2.\texttt{clr}$$

- We can prove

$$\langle \Sigma'_1, \texttt{Lb}\ l_1\ e''_1 \rangle \approx_L \langle \Sigma'_2, \texttt{Lb}\ l_2\ e''_2 \rangle$$

- by just case analysis if $\Sigma'_1.\texttt{lbl} \sqsubseteq L$ and applying the definition of low-equivalence for configurations

Final Remarks

- We formalize the ideas behind LIO
  - Language: simple call-by-name lambda-calculus
- Semantics
  - Security checks
- Types (not very interesting)
- Simulation
- Low-equivalence
- Non-interference theorem