IFC Inside: A General Approach to Retrofitting Languages with Dynamic Information Flow Control

Stefan Heule, Deian Stefan, Edward Z. Yang, John C. Mitchell, Alejandro Russo

Stanford University, Chalmers University

What is IFC?

- Not all information in a program is equal
  - Some might be more sensitive
- Information flow control …
  - … *tracks* where information flows
  - … allows *policies to restrict* flows of information

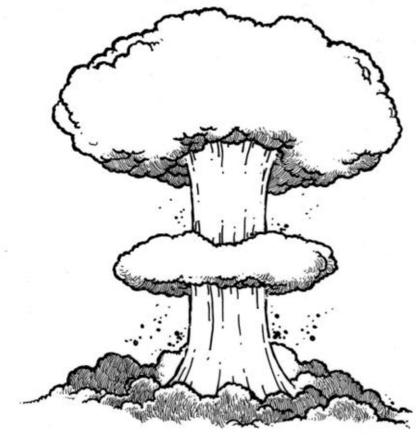
Example: security on the web

- Modern web pages involve many components
  - Javascript of the website
  - Javascript of some (untrusted) library
  - Advertising code
  - Browser addons

- Web content can be very sensitive
  - Online banking, passwords, personal information, etc.

Current situation

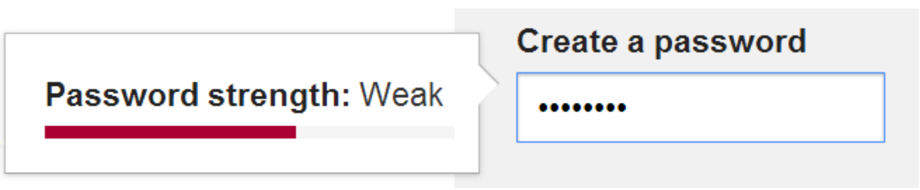
- Code written by many different parties
  - Potentially mutually distrusting parties
  - Computing over sensitive data



- IFC to the rescue
  - Label sensitive data as such
  - Prevent flow of sensitive data to undesired places (like arbitrary web servers)

Concrete Example

- Password strength checking



- Website uses `check_strength(pw)` from a (not fully trusted) library
  - The library could send the password to `bad.com`
- With IFC, we can precisely decide what the library can do with the password

Why isn't everyone using IFC?

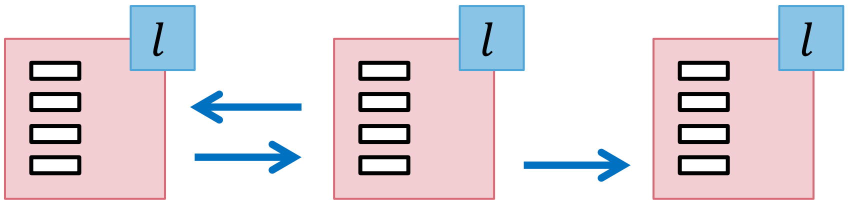
- We are interested in a *dynamic* IFC system

- A key challenge is performance
  - Tracking information at a fine-grained level is expensive

Coarse-grained IFC

- The program is split into computational units (tasks)
  - All data within one task has a single label

- Different computational units can communicate



Advantages of coarse-grained IFC

1. Efficiency
   - Checks only at isolation boundaries
2. Minimal changes to language
   - If added to a language retroactively
3. Reuse existing programs
4. Simple
   - to understand and reason about

Goals

- Formally define a core calculus for a coarse-grained dynamic IFC system
- Combine IFC language with *any* programming language
- Prove security guarantees of IFC (known as non-interference)

Approach Overview

- Given a target language (any programming language)
- Define an IFC language
  - Minimal calculus, only IFC features
- Combine target and IFC language
  - Allow target language to call into IFC, and vice-versa
- Careful definition of the IFC language allows the overall system to provide isolation, regardless of what the target language does

IFC language

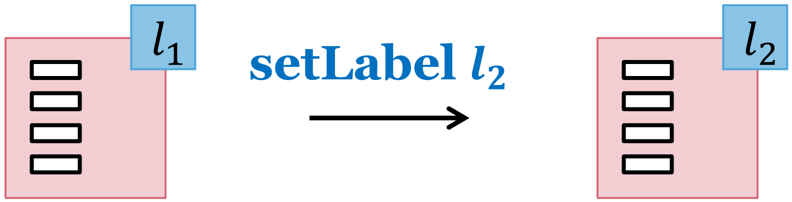
- Tag data with security labels
  - Labels form a lattice, and determine how data can flow inside an application

- Example lattice
  - Two labels *H* (high) and *L* (low)
  - Flow from *H* to *L* is not allowed

$$\begin{array}{c} H \\ \uparrow \\ L \end{array}$$

IFC language: labels

- Get and set the current label
  - **setLabel**, **getLabel**

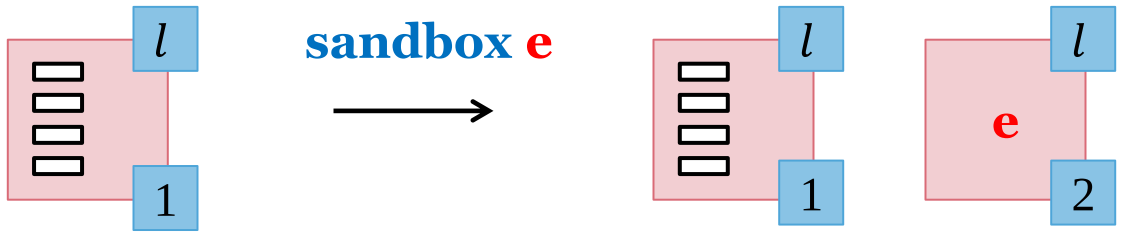


- Setting the label is only allowed to *raise* the label

- Can also compute on labels
  - $\sqsubseteq, \sqcap, \sqcup$

IFC language: sandboxing

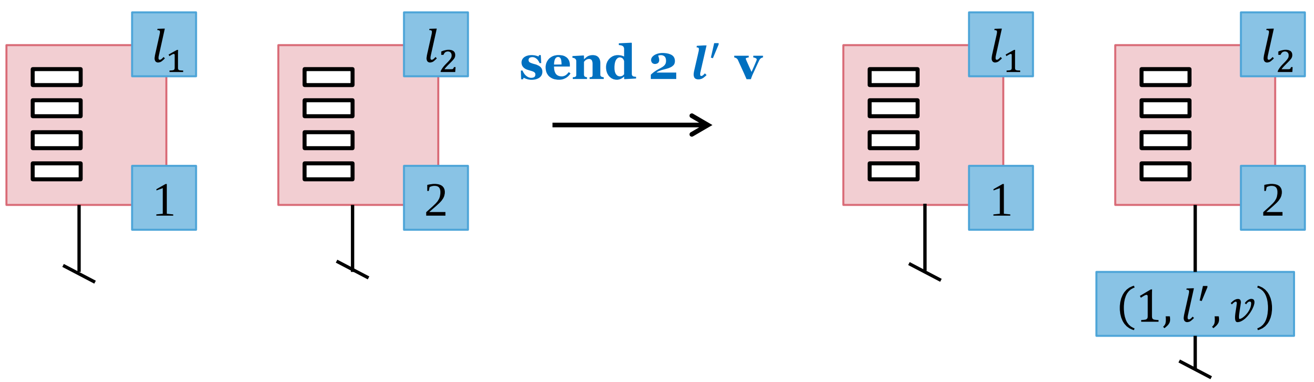
- Isolate an expression as a new task
  - **sandbox e**



Inter-task communication

- Tasks can send and receive messages

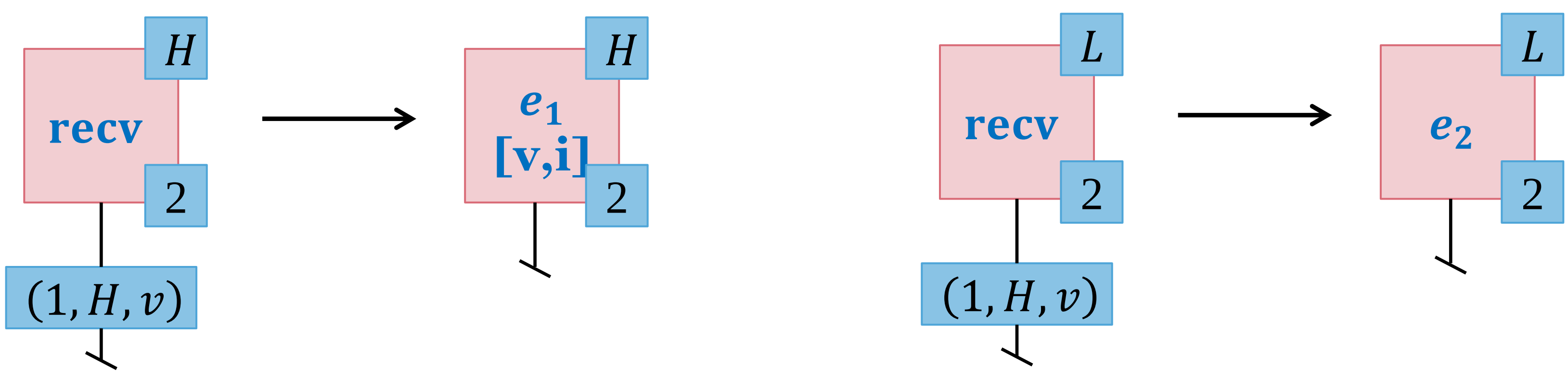
- Send message **v** to task **i**, protected by label $\boldsymbol{l'}$
  - **send i $\boldsymbol{l'}$ v**
  - Can only send messages at or above current label



Inter-task communication

- Receiving either binds a message **v** and sender **i** in $\boldsymbol{e_1}$, or execution continues in $\boldsymbol{e_2}$ (if there is no message)
  - Messages that are above the current level are never received

recv i,v in $\boldsymbol{e_1}$ else $\boldsymbol{e_2}$



Formal treatment

What is a programming language?

- Need a formal definition of a language
  - Global store **Σ**
  - Evaluation context **E**
  - Expression syntax **e**, some expressions are values **v**
  - Reduction relation $\rightarrow$

- This is the **target language**

Example: Mini-ECMAScript

$$\begin{aligned}
\mathbf{v} &::= \lambda \mathbf{x}.\mathbf{e} \mid \mathbf{true} \mid \mathbf{false} \mid \mathbf{a} \\
\mathbf{e} &::= \mathbf{v} \mid \mathbf{x} \mid \mathbf{e}\ \mathbf{e} \mid \mathbf{if}\ \mathbf{e}\ \mathbf{then}\ \mathbf{e}\ \mathbf{else}\ \mathbf{e} \\
&\quad \mid \mathbf{ref}\ \mathbf{e} \mid !\mathbf{e} \mid \mathbf{e} := \mathbf{e} \mid \mathbf{fix}\ \mathbf{e} \\
\mathbf{E} &::= [\cdot]_{\mathbf{T}} \mid \mathbf{E}\ \mathbf{e} \mid \mathbf{v}\ \mathbf{E} \mid \mathbf{if}\ \mathbf{E}\ \mathbf{then}\ \mathbf{e}\ \mathbf{else}\ \mathbf{e} \\
&\quad \mid \mathbf{ref}\ \mathbf{E} \mid !\mathbf{E} \mid \mathbf{E} := \mathbf{e} \mid \mathbf{v} := \mathbf{E} \mid \mathbf{fix}\ \mathbf{E}
\end{aligned}$$

T-APP

$$\frac{}{\mathcal{E}_{\mathbf{\Sigma}}\,[(\lambda x.\mathbf{e})\ \mathbf{v}] \to \mathcal{E}_{\mathbf{\Sigma}}\,[\{\mathbf{v}\ /\ x\}\ \mathbf{e}]}$$

T-IFTRUE

$$\frac{}{\mathcal{E}_{\mathbf{\Sigma}}\,[\ \mathbf{if}\ \mathbf{e}_2] \to \mathcal{E}_{\mathbf{\Sigma}}\,[\mathbf{e}_1]}$$

T-IFFALSE

$$\frac{}{\mathcal{E}_{\mathbf{\Sigma}}\,[\ \mathbf{if\ false\ then}\ \mathbf{e}_1\ \mathbf{else}\ \mathbf{e}_2] \to \mathcal{E}_{\mathbf{\Sigma}}\,[\mathbf{e}_2]}$$

T-REF

$$\frac{\mathrm{fresh}(\mathbf{a})}{\mathcal{E}_{\mathbf{\Sigma}}\,[\mathbf{ref}\ \mathbf{v}] \to \mathcal{E}_{\mathbf{\Sigma}[\mathbf{a}\mapsto\mathbf{v}]}\,[\mathbf{a}]}$$

T-DEREF

$$\frac{(\mathbf{a},\mathbf{v}) \in \mathbf{\Sigma}}{\mathcal{E}_{\mathbf{\Sigma}}\,[!\mathbf{a}] \to \mathcal{E}_{\mathbf{\Sigma}}\,[\mathbf{v}]}$$

T-ASS

$$\frac{}{\mathcal{E}_{\mathbf{\Sigma}}\,[\mathbf{a} := \mathbf{v}] \to \mathcal{E}_{\mathbf{\Sigma}[\mathbf{a}\mapsto\mathbf{v}]}\,[\mathbf{v}]}$$

T-FIX

$$\frac{}{\mathcal{E}_{\mathbf{\Sigma}}\,[\mathbf{fix}\ (\lambda x.e)] \to \mathcal{E}_{\mathbf{\Sigma}}\,[\{\mathbf{fix}\ (\lambda x.e)\ /\ x\}\ e]}$$

Notation

- Rules are standard, except we use $\mathcal{E}_{\Sigma}$ instead of normal context $\mathbf{E}$

$$\frac{\text{T-IFFALSE}}{\mathcal{E}_{\boldsymbol{\Sigma}}\,[\,\textbf{if false then } \mathbf{e}_1 \textbf{ else } \mathbf{e}_2] \rightarrow \mathcal{E}_{\boldsymbol{\Sigma}}\,[\mathbf{e}_2]}$$

- Obtain normal semantics with

$$\mathcal{E}_{\boldsymbol{\Sigma}}\,[\mathbf{e}] \triangleq \boldsymbol{\Sigma}, \mathbf{E}\,[\mathbf{e}]$$

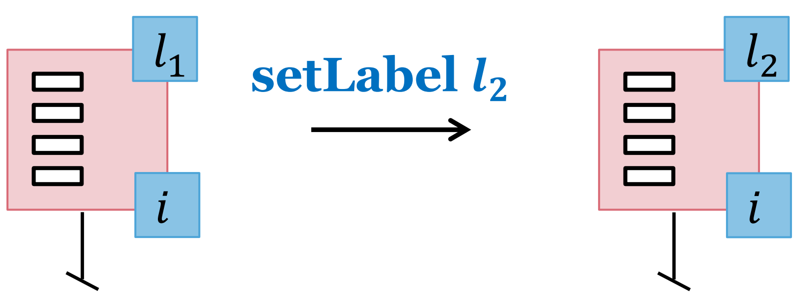
- Later, we re-interpret what $\mathcal{E}$ stands for

IFC language

- Also defined in terms of an special $\mathcal{E}$

I-SETLABEL

$$\frac{l \sqsubseteq l'}{\mathcal{E}^{i,l}_{\Sigma}\left[\mathbf{setLabel}\ l'\right] \to \mathcal{E}^{i,l'}_{\Sigma}\left[\langle\rangle\right]}$$



Embedding [Matthews and Findler, POPL'07]

- Extend IFC and target language syntax

$$
\begin{aligned}
v &::= \cdots \mid {}^{\mathrm{IT}}\lceil \mathbf{v} \rceil \\
e &::= \cdots \mid {}^{\mathrm{IT}}\lceil \mathbf{e} \rceil \\
E &::= \cdots \mid {}^{\mathrm{IT}}\lceil \mathbf{E} \rceil
\end{aligned}
\qquad
\begin{aligned}
\mathbf{v} &::= \cdots \mid {}_{\mathrm{TI}}\lfloor v \rfloor \\
\mathbf{e} &::= \cdots \mid {}_{\mathrm{TI}}\lfloor e \rfloor \\
\mathbf{E} &::= \cdots \mid {}_{\mathrm{TI}}\lfloor E \rfloor
\end{aligned}
$$

- Re-interpret context and reduction relation

$$
\begin{aligned}
&\mathcal{E}_{\mathbf{\Sigma}}\,[\mathbf{e}] \triangleq \Sigma; \langle \mathbf{\Sigma}, E[\mathbf{e}]_{\mathbf{T}} \rangle_{l}^{i}, \ldots \\
&\mathcal{E}_{\Sigma}^{i,l}\,[e] \triangleq \Sigma; \langle \mathbf{\Sigma}, E[e]_{I} \rangle_{l}^{i}, \ldots
\end{aligned}
$$

Sandboxing

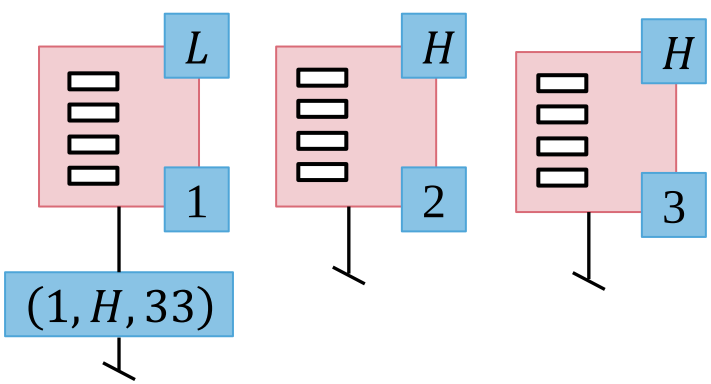
I-SANDBOX

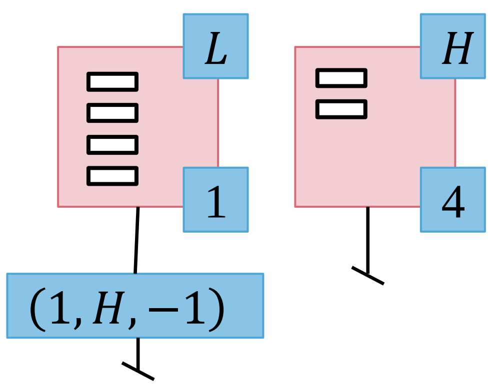
$$\frac{\Sigma' = \Sigma\left[i' \mapsto \epsilon\right] \qquad \boldsymbol{\Sigma}' = \kappa\left(\boldsymbol{\Sigma}\right) \\ t_1 = \langle \boldsymbol{\Sigma}, E[i']\rangle_l^i \qquad t_{\text{new}} = \langle \boldsymbol{\Sigma}', e\rangle_l^{i'} \qquad \text{fresh}(i')}{\Sigma; \langle \boldsymbol{\Sigma}, E[\textbf{sandbox}\ e]_I\rangle_l^i, \ldots \overset{\alpha}{\hookrightarrow} \Sigma'; \alpha_{\text{sandbox}}(t_1, \ldots, t_{\text{new}})}$$

- **sandbox e** does
  - Create new task for e
  - Schedule e according to scheduling policy

Security Guarantees

- Non-interference:
  - Intuitively: An attacker that can only see values up to level l should not see a difference in behavior if values at level $l' > l$ are changed





Erasure function

- Formally, we need an erasure function ε_l
  - Erases all data above l to ■
  - Program c_1 and c_2 are l-equivalent, $c_1 \approx_l c_2$, iff $\varepsilon_l(c_1) = \varepsilon_l(c_2)$

- For our system, ε_l erases the following:
  - Any tasks with current label above l
  - Any messages with label above l

Termination sensitive non-interference (TSNI)

For all programs c_1, c_2, c_1' and labels l, such that

$$c_1 \approx_l c_2 \qquad \text{and} \qquad c_1 \hookrightarrow^* c_1'$$

then there exists c_2' such that

$$c_1' \approx_l c_2' \qquad \text{and} \qquad c_2 \hookrightarrow^* c_2'$$

Theorem: Any target language combined with our IFC language with round robin scheduling satisfies TSNI.

Real world examples

- Is this actually practical?

- One challenge are external effects
  - File system, internet connection, etc.

- Possible solutions
  - Make external effects inaccessible
  - Internalize them into the IFC language
    - Labeled file system

SWAPI

- Implementation of coarse-grained dynamic IFC system for Javascript
- Websites have access to XHR constructor
  - XHR requests need to be modeled at the IFC language level
- SWAPI chooses origins (`example.com`) as labels

Password strength checker in SWAPI

- Even if `check_strength(pw)` is completely untrusted, it cannot send the password to `bad.com`

- Execute `check_strength` in a sandboxed task

Conclusions

- Coarse-grained IFC is great
  - Allows for language-independent IFC system
  - Efficient, yet flexible

- Combining operational semantics of two languages as key mechanism to formalize our system

Thank you.

Questions?