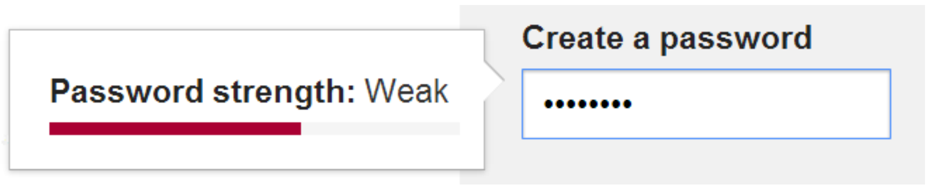
IFC Inside: Retrofitting Languages with Dynamic Information Flow Control

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

Stanford University, Chalmers University

Motivating Example: Web Security

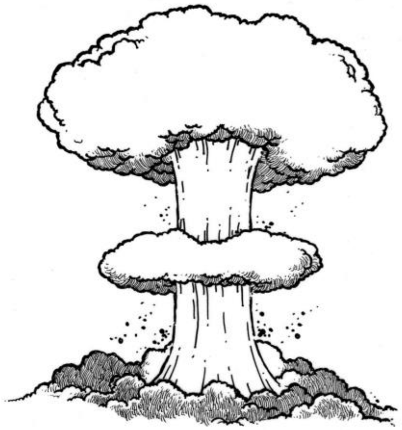


- Website uses `check_strength(pw)` from some library
  - Danger: the library could send the password to `bad.com`
  - Website author has little control over this

[Van Acker et al., CODASPY'15]

Web Security Today

- Code written by many different parties
  - Potentially mutually distrusting parties (website code, utility/framework libraries, advertising code, …)
  - Computing over sensitive data (passwords, healthcare information, banking data)



Possible Solution: IFC

- Information flow control …
  - … *tracks* where information flows
  - … allows *policies to restrict* flows of information

- In the example
  - Label password as sensitive
  - Restrict its dissemination (e.g. to arbitrary webservers)

What kind of IFC?

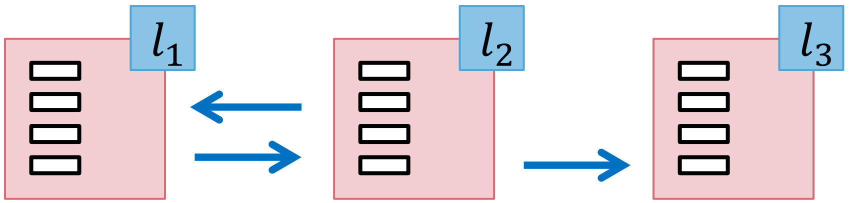
- Various trade-offs in IFC systems
  - Dynamic vs static
  - What kind of labels
  - Granularity at with information is tracked

- Sweetspot: dynamic, coarse-grained IFC

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



This Talk

- Given an existing programming language, how can we add dynamic IFC?
- Minimal changes to language
  - Simplifies implementation
- Formal security guarantees

Approach Overview

- Given a target language
  - Any programming language for which we can control external effects
- 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 tasks 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

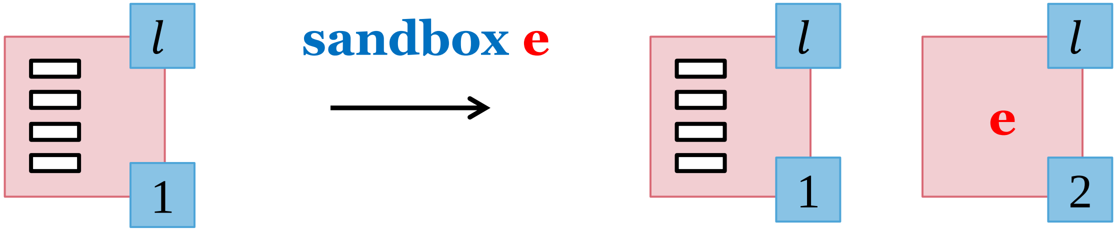
- Get and set the current label
  - **setLabel**, **getLabel**

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

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

IFC language: sandboxing

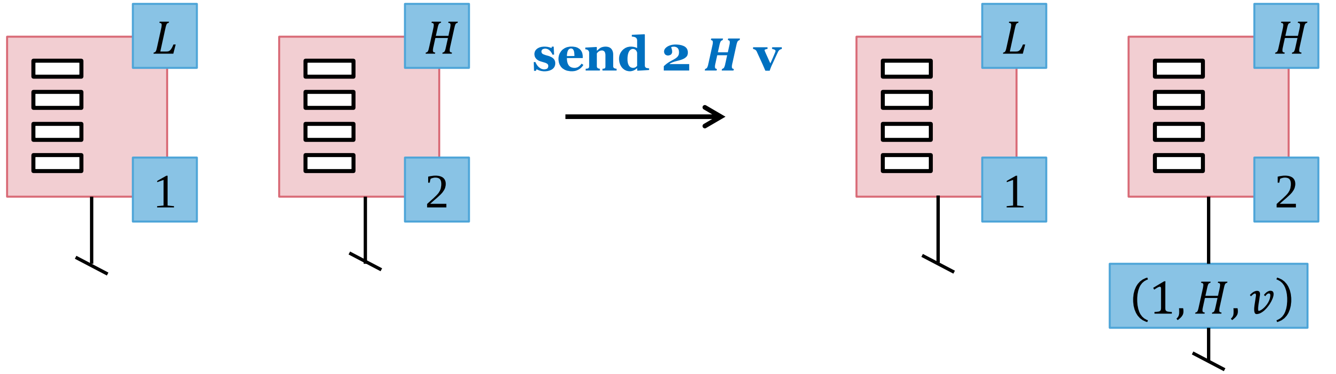
- Isolate an expression as a new task
  - **sandbox e**



- New task has separate state

Inter-task communication

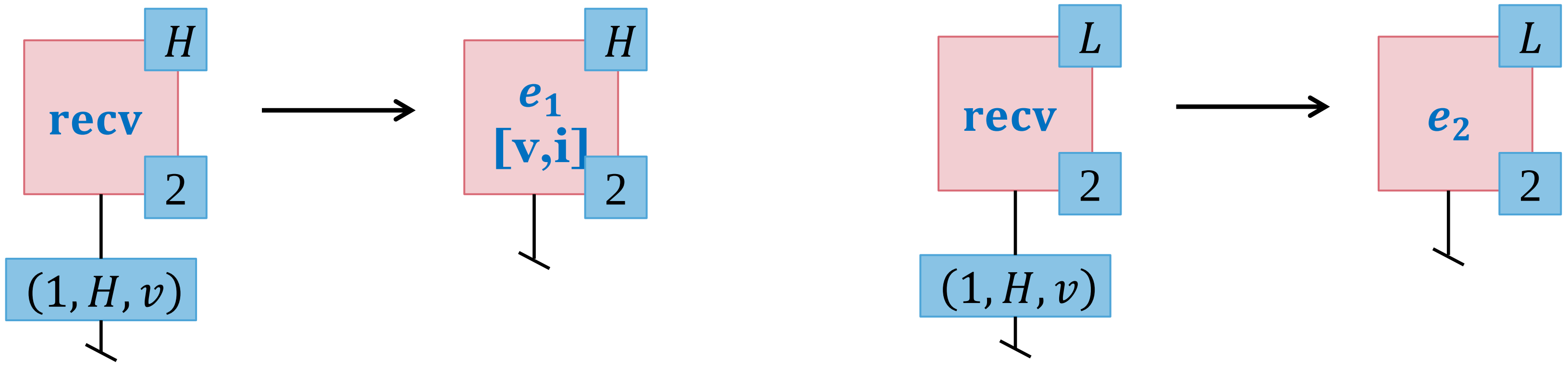
- Tasks can send and receive messages
- Send message **v** to task **i**, protected by label ***l***
  - **send i *l* v**
  - Can only send messages at or above current label



Inter-task communication

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

recv i,v in e_1 else 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 →

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

T-IFTRUE
$$\frac{}{\mathcal{E}_{\boldsymbol{\Sigma}}\,[\ \mathbf{if}\ \mathbf{e}_2] \rightarrow \mathcal{E}_{\boldsymbol{\Sigma}}\,[\mathbf{e}_1]}$$

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

T-REF
$$\frac{\mathrm{fresh}(\mathbf{a})}{\mathcal{E}_{\boldsymbol{\Sigma}}\,[\mathbf{ref}\ \mathbf{v}] \rightarrow \mathcal{E}_{\boldsymbol{\Sigma}[\mathbf{a}\mapsto\mathbf{v}]}\,[\mathbf{a}]}$$

T-DEREF
$$\frac{(\mathbf{a}, \mathbf{v}) \in \boldsymbol{\Sigma}}{\mathcal{E}_{\boldsymbol{\Sigma}}\,[!\mathbf{a}] \rightarrow \mathcal{E}_{\boldsymbol{\Sigma}}\,[\mathbf{v}]}$$

T-ASS
$$\frac{}{\mathcal{E}_{\boldsymbol{\Sigma}}\,[\mathbf{a} := \mathbf{v}] \rightarrow \mathcal{E}_{\boldsymbol{\Sigma}[\mathbf{a}\mapsto\mathbf{v}]}\,[\mathbf{v}]}$$

T-FIX
$$\frac{}{\mathcal{E}_{\boldsymbol{\Sigma}}\,[\mathbf{fix}\ (\lambda x.e)] \rightarrow \mathcal{E}_{\boldsymbol{\Sigma}}\,[\{\mathbf{fix}\ (\lambda x.e)\ /\ x\}\ e]}$$

Notation

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

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

- Obtain normal semantics with

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

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

IFC language

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

I-SETLABEL

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

Embedding [Matthews and Findler, POPL'07]

- Extend IFC and target language syntax

$$\mathbf{e} ::= \cdots \mid {}_{\mathrm{TI}}\lfloor e \rfloor$$

$$e ::= \cdots \mid {}^{\mathrm{IT}}\lceil \mathbf{e} \rceil$$

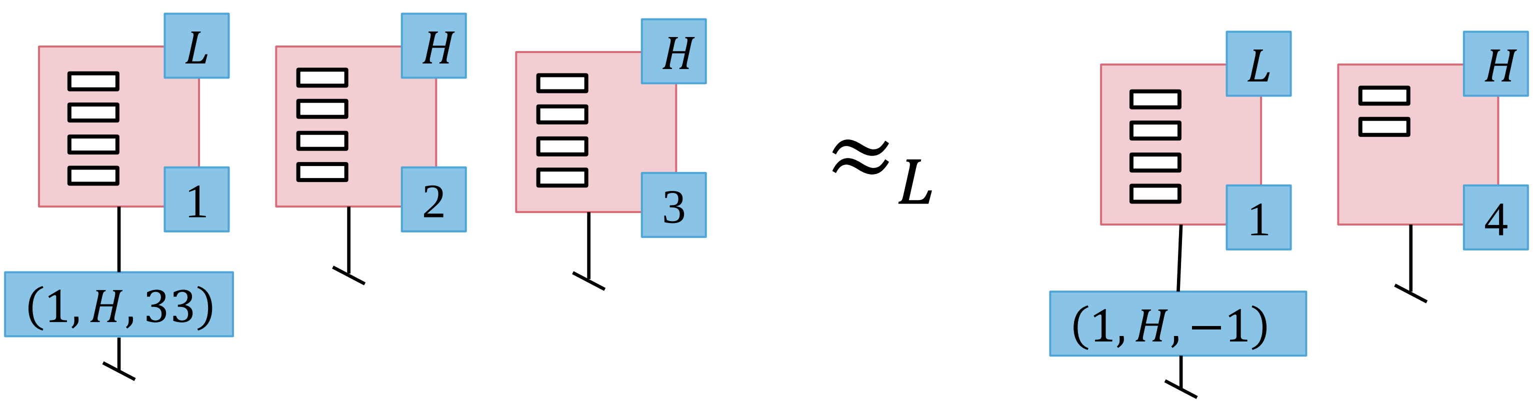
- Re-interpret context and reduction relation

$$\mathcal{E}_{\boldsymbol{\Sigma}}[\mathbf{e}] \triangleq \Sigma; \langle \boldsymbol{\Sigma}, E[\mathbf{e}]_{\mathrm{T}} \rangle_l^i, \ldots$$

$$\mathcal{E}_{\Sigma}^{i,l}[e] \triangleq \Sigma; \langle \boldsymbol{\Sigma}, E[e]_I \rangle_l^i, \ldots$$

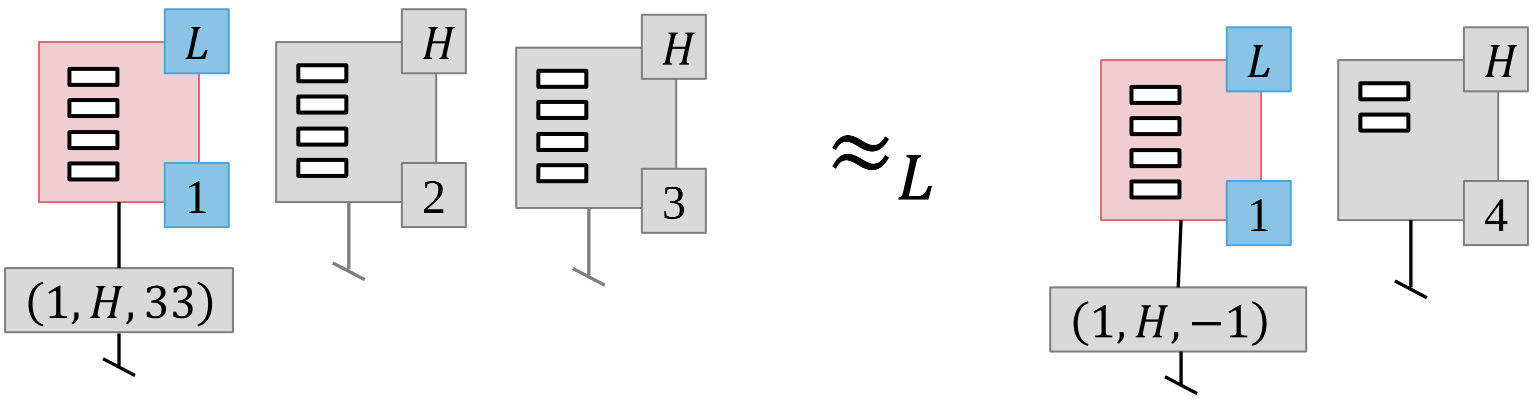
Security Guarantees

- 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



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

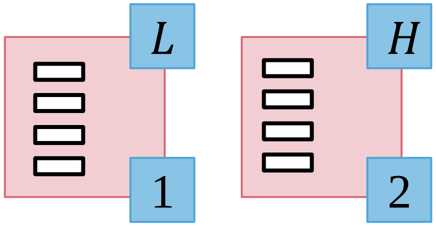
$$c_1' \approx_l c_2' \qquad \text{and} \qquad c_2 \hookrightarrow^* c_2'$$

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

Practicality

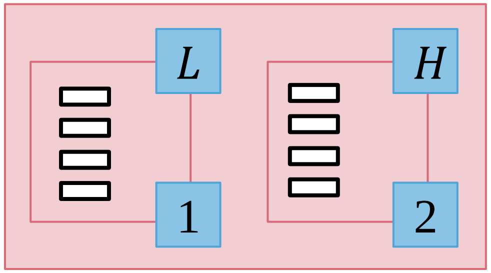
- Formalism requires separate heaps

$$\Sigma; \langle \boldsymbol{\Sigma}_1, e_1 \rangle_{l_1}^{i_1}, \langle \boldsymbol{\Sigma}_2, e_2 \rangle_{l_2}^{i_2} \ \ldots$$



- An implementation might want to have one heap

$$\Sigma; \boldsymbol{\Sigma}; \langle e_1 \rangle_{l_1}^{i_1}, \langle e_2 \rangle_{l_2}^{i_2}, \ldots$$



- Naïve implementation is insecure
    - Shared references, need additional checks

Modifying the Combined Language

- Single heap only requires restricting transition rules
  - Intuitively appears OK
  - In general, not safe

I-SEND

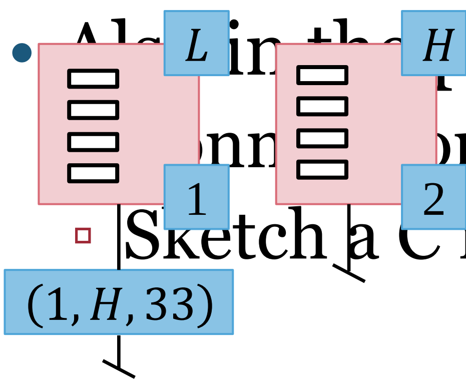
$$\frac{l \sqsubseteq l' \qquad \Sigma(i') = \Theta \qquad \Sigma' = \Sigma\left[i' \mapsto (l', i, v), \Theta\right] \qquad v \text{ not } \mathbf{ref}}{\mathcal{E}_{\Sigma}^{i,l}\left[\mathbf{send}\ i'\ l'\ v\right] \rightarrow \mathcal{E}_{\Sigma'}^{i,l}\left[\langle\rangle\right]}$$

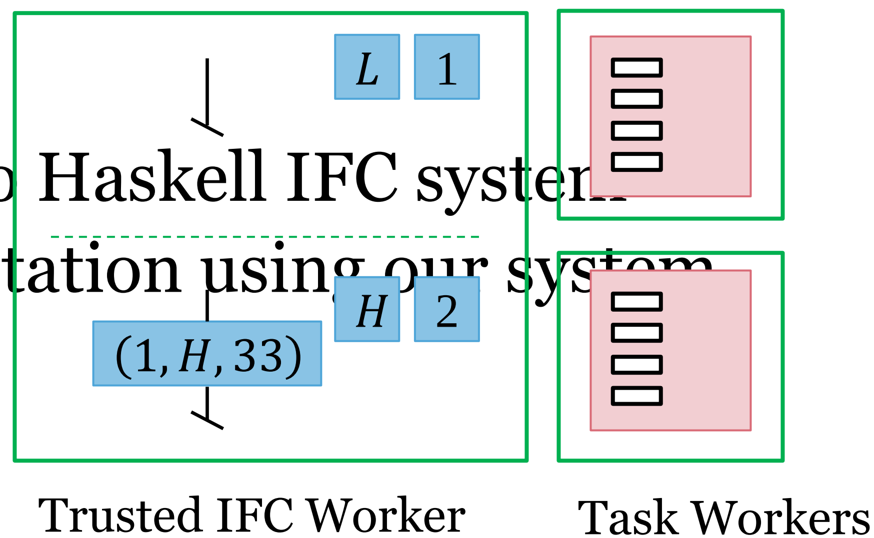
- We give a class of restrictions that is safe
  - In a nutshell: restriction cannot depend on secret data

Implementation

- IFC for Node.js
  - No changes to Javascript runtime or Node.js
  - Worker threads implement tasks
  - Trusted main worker implements IFC checks

- Also in the paper:
  - Connect formalism to Haskell IFC system
  - Sketch a C implementation using our system





Conclusions

- Formalism for dynamic coarse-grained IFC for many programming languages
  - Little reliance on language details
- Combining operational semantics of two languages as key mechanism to formalize our system
  - Allows security proofs to be once and for all

Thank you.

Questions?