An Information Flow Calculus for Non-Interference

Clément Aubert and **Neea Rusch**
Augusta University, United States

The 19th Workshop on Programming Languages
and Analysis for Security (PLAS 2024)

14 October 2024

Motivation: Guarantee Runtime Properties

Idea: use *programming languages* to guarantee runtime properties
⇒ what(ever) can be expressed is known to be satisfactory

. . . by applying techniques from *implicit computational complexity*.

Implicit Computational Complexity (ICC)

Let L be a programming language, C a complexity class, and $[\![p]\!]$ the function computed by program p.

Find a restriction $R \subseteq L$, such that the following equality holds:

$$\{[\![p]\!] \mid p \in R\} = C$$

The variables L, C, and R are the parameters that vary greatly between different ICC systems[1].

---

[1] Romain Péchoux. *Complexité implicite : bilan et perspectives*. Habilitation à Diriger des Recherches (HDR). 2020. URL: `https://hal.univ-lorraine.fr/tel-02978986`.

Characteristics of ICC Techniques

- Many advantaged for performing program analysis
  e.g., static, automatic, compositional, sound guarantees
- Adjustable techniques with representational strengths
  e.g., require little structure, bypass difficulties, program abstractions for free
- Trade guarantees for precision and expressive power:
  Approximative results, limited syntax

Applications

Guaranteeing resource usage:

☑ Static analysis of complexity

Analyzing and guaranteeing *other semantic properties*:

☑ Compiler optimizations
☑ Invariant inference

☐ Security properties?

Implicit Complexity Meets Security

SAFE programs[2]
Security type system modified to track data-size increase
$\Rightarrow$ characterization of polynomial time functions.

Stratified programs[3]
Security type system + heap memory restriction + shape analysis
$\Rightarrow$ more expressive characterization of P-time functions.

---

[2] Jean-Yves Marion. "A Type System for Complexity Flow Analysis". In: *2011 IEEE 26th Annual Symposium on Logic in Computer Science*. 2011, pp. 123–132. DOI: `10.1109/LICS.2011.41`.

[3] Emmanuel Hainry and Romain Péchoux. "A General Noninterference Policy for Polynomial Time". In: *Proc. ACM Program. Lang.* 7.POPL (2023), pp. 806–832. DOI: `10.1145/3571221`.

Implicit Complexity Meets Security

security $\longrightarrow$ implicit complexity

Implicit Complexity Meets Security

$$\text{security} \xleftrightarrow{?} \text{implicit complexity}$$

TODO: Plan

1. Define programming language and semantics
2. Take an ICC-based data-flow calculus
3. Adjust calculus to track a security property (non-interference)
4. Show useful applications

Imperative Language

$$
\begin{aligned}
\mathit{var} &::= \texttt{i} \mid \dots \mid \texttt{t} \mid \dots \mid \texttt{x}_1 \mid \dots \mid \mathit{var}[\mathit{exp}] && \text{(Variable)}\\
\mathit{exp} &::= \mathit{var} \mid \mathit{val} \mid \mathit{op}(\mathit{exp}, \dots, \mathit{exp}) && \text{(Expression)}\\
\mathit{com} &::= \mathit{var} \leftarrow \mathit{exp} \mid \mathbf{\texttt{skip}} \mid \\
&\quad\ \mathbf{\texttt{if}}\ \mathit{exp}\ \mathbf{\texttt{then}}\ \mathit{com}\ \mathbf{\texttt{else}}\ \mathit{com} \mid \\
&\quad\ \mathbf{\texttt{while}}\ \mathit{exp}\ \mathbf{\texttt{do}}\ \mathit{com} \mid \mathit{com};\mathit{com} && \text{(Command)}
\end{aligned}
$$

Semantics as expected from syntax; a *program* is a sequence of commands.

Non-interfering Programs

Definition: Non-interference

We let SC be an *information flow policy* lattice, and ℓ the level assignment that assigns to each variable $\mathbf{x}$ its security class (or *level*) $\ell(\mathtt{x}) \in \mathrm{SC}$. A command $\mathtt{C}$ is *non-interfering for* ℓ if for all level $l \in \mathrm{SC}$, and all variable values lists $\vec{v_1}$ and $\vec{v_2}$,

$$\vec{v_1} =_l^{\ell\leqslant} \vec{v_2}, \mathtt{C}[\vec{v_1} \to \vec{v_1'}], \mathtt{C}[\vec{v_2} \to \vec{v_2'}] \implies \vec{v_1'} =_l^{\ell\leqslant} \vec{v_2'}$$

Informally: changing the value received by higher-level variables does not impact the values of lower-level variables at any program state.

Two values lists $\vec{v}$ and $\vec{v'}$ are *up-to* l *equivalent* $\vec{v} =^{\ell}_{l\leqslant} \vec{v'}$ iff $\ell(\mathtt{x}_i) \leqslant l \implies v_i = v'_i$, and $\mathtt{C}[\vec{v} \to \vec{v'}]$ means $\mathtt{C}[\vec{v}]$ terminates and after executing all the commands in $\mathtt{C}[\vec{v}]$, $\mathtt{x}_i$ contains the value v'_i, for $1 \leqslant i \leqslant n$.

Information Flow Calculus

We track data-flow dependencies between variables in expressions and commands; tagging them as *modified by* (out), *used by* (in), or *occurring* (occ).

The variables occurring in expression e:

$$\mathrm{Occ}(\mathtt{x}) = \mathtt{x} \qquad \mathrm{Occ}(\mathtt{t[e]}) = \mathtt{t} \cup \mathrm{Occ}(\mathtt{e})$$

$$\mathrm{Occ}(\mathit{val}) = \emptyset \qquad \mathrm{Occ}(\mathtt{op(e_1, \ldots, e_n)}) = \mathrm{Occ}(\mathtt{e}_1) \cup \cdots \cup \mathrm{Occ}(\mathtt{e_n})$$

Information Flow Calculus

Command $\mathtt{C}$	**Out** $(\mathtt{C})$	**In** $(\mathtt{C})$	**Occ** $(\mathtt{C})$ = **Out** $(\mathtt{C}) \cup$ **In** $(\mathtt{C})$
$\mathtt{x{=}e}$	$\mathtt{x}$	$\mathrm{Occ}(\mathtt{e})$	$\mathtt{x} \cup \mathrm{Occ}(\mathtt{e})$
$\mathtt{t[e_1]{=}e_2}$	$\mathtt{t}$	$\mathrm{Occ}(\mathtt{e_1}) \cup \mathrm{Occ}(\mathtt{e_2})$	$\mathtt{t} \cup \mathrm{Occ}(\mathtt{e_1}) \cup \mathrm{Occ}(\mathtt{e_2})$
skip	$\emptyset$	$\emptyset$	$\emptyset$
if $\mathtt{e}$ **then** $\mathtt{C_1}$ **else** $\mathtt{C_2}$	$\mathrm{Out}(\mathtt{C_1}) \cup \mathrm{Out}(\mathtt{C_2})$	$\mathrm{Occ}(\mathtt{e}) \cup \mathrm{In}(\mathtt{C_1}) \cup \mathrm{In}(\mathtt{C_2})$	$\mathrm{Occ}(\mathtt{e}) \cup \mathrm{Occ}(\mathtt{C_1}) \cup \mathrm{Occ}(\mathtt{C_2})$
while $\mathtt{e}$ **do** $\mathtt{C}$	$\mathrm{Out}(\mathtt{C})$	$\mathrm{Occ}(\mathtt{e}) \cup \mathrm{In}(\mathtt{C})$	$\mathrm{Occ}(\mathtt{e}) \cup \mathrm{Occ}(\mathtt{C})$
$\mathtt{C_1; C_2}$	$\mathrm{Out}(\mathtt{C_1}) \cup \mathrm{Out}(\mathtt{C_2})$	$\mathrm{In}(\mathtt{C_1}) \cup \mathrm{In}(\mathtt{C_2})$	$\mathrm{Occ}(\mathtt{C_1}) \cup \mathrm{Occ}(\mathtt{C_2})$

The set of variables modified by (resp. used by, occurring in) command $\mathtt{C}$.

Security Flow Matrix (SFM)

Given command `C` with n variables, a *security flow matrix*, $\mathbb{M}(\mathtt{C})$, is an $n \times n$ matrix of coefficients ($\cdot$, 💧) tracking information flow.

$$\begin{array}{cc} & \begin{array}{ccc} \cdots & \mathtt{y} & \cdots \end{array} \\ \begin{array}{c} \vdots \\ \mathtt{x} \\ \vdots \end{array} & \begin{pmatrix} \ddots & & \iddots \\ & \text{💧} & \\ \iddots & & \ddots \end{pmatrix} \end{array}$$

$\mathbb{M}(\mathtt{C})(\mathbf{x}, \mathbf{y})$ denotes the coefficient at **row** $\mathtt{x}$ and **column** $\mathtt{y}$.

`C` has a *violation* if there exists $\mathbf{x}$ and $\mathbf{y}$ such that $\mathbb{M}(\mathtt{C})(\mathbf{x}, \mathbf{y}) =$ 💧 and $\ell(\mathtt{y}) < \ell(\mathtt{x})$.

Security Flow Matrix: Examples

$\mathtt{C}$	**Out** $(\mathtt{C})$, **In** $(\mathtt{C})$	$\mathbb{M}(\mathtt{C})$	**Violation(s)**
$\mathtt{w} = 3$	$\mathrm{Out}(\mathtt{C}) = \{\mathtt{w}\}$ $\mathrm{In}(\mathtt{C}) = \emptyset$	$\begin{array}{cc} & \mathtt{w} \\ \mathtt{w} & \begin{pmatrix} \cdot \end{pmatrix} \end{array}$	None
$\mathtt{x} = \mathtt{y}$	$\mathrm{Out}(\mathtt{C}) = \{\mathtt{x}\}$ $\mathrm{In}(\mathtt{C}) = \{\mathtt{y}\}$	$\begin{array}{cc} & \begin{matrix} \mathtt{x} & \mathtt{y} \end{matrix} \\ \begin{matrix} \mathtt{x} \\ \mathtt{y} \end{matrix} & \begin{pmatrix} \cdot & \cdot \\ \text{💧} & \cdot \end{pmatrix} \end{array}$	If $\ell(\mathtt{x}) < \ell(\mathtt{y})$
$\mathtt{w} = \mathtt{t}[\mathtt{x} + 1]$	$\mathrm{Out}(\mathtt{C}) = \{\mathtt{w}\}$ $\mathrm{In}(\mathtt{C}) = \{\mathtt{t}, \mathtt{x}\}$	$\begin{array}{cc} & \begin{matrix} \mathtt{w} & \mathtt{t} & \mathtt{x} \end{matrix} \\ \begin{matrix} \mathtt{w} \\ \mathtt{t} \\ \mathtt{x} \end{matrix} & \begin{pmatrix} \cdot & \cdot & \cdot \\ \text{💧} & \cdot & \cdot \\ \text{💧} & \cdot & \cdot \end{pmatrix} \end{array}$	If $\ell(\mathtt{w}) < \ell(\mathtt{t})$ or $\ell(\mathtt{w}) < \ell(\mathtt{x})$
$\mathtt{t}[\mathtt{i}] = \mathtt{u} + \mathtt{j}$	$\mathrm{Out}(\mathtt{C}) = \{\mathtt{t}\}$ $\mathrm{In}(\mathtt{C}) = \{\mathtt{i}, \mathtt{u}, \mathtt{j}\}$	$\begin{array}{cc} & \begin{matrix} \mathtt{t} & \mathtt{i} & \mathtt{u} & \mathtt{j} \end{matrix} \\ \begin{matrix} \mathtt{t} \\ \mathtt{i} \\ \mathtt{u} \\ \mathtt{j} \end{matrix} & \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \text{💧} & \cdot & \cdot & \cdot \\ \text{💧} & \cdot & \cdot & \cdot \\ \text{💧} & \cdot & \cdot & \cdot \end{pmatrix} \end{array}$	If $\ell(\mathtt{t}) < \ell(\mathtt{i})$, or $\ell(\mathtt{t}) < \ell(\mathtt{u})$, or $\ell(\mathtt{t}) < \ell(\mathtt{j})$.

Derivation Example I

```
if (h==0) then y=1 else skip;   // C1
if (y==0) then z=1 else y=z     // C2
```

$$\dfrac{\dfrac{\dfrac{}{\texttt{h==0} : \begin{pmatrix}\cdot&\bullet&\cdot\\\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\end{pmatrix}}\,\text{Cr}}{\texttt{C1} : \begin{pmatrix}\cdot&\bullet&\cdot\\\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\end{pmatrix}}\,\text{Cond} \qquad \dfrac{\dfrac{}{\texttt{y==0} : \begin{pmatrix}\cdot&\cdot&\cdot\\\cdot&\cdot&\bullet\\\cdot&\cdot&\cdot\end{pmatrix}}\,\text{Cr} \quad \dfrac{}{\texttt{y=z} : \begin{pmatrix}\cdot&\cdot&\cdot\\\cdot&\cdot&\cdot\\\cdot&\bullet&\cdot\end{pmatrix}}\,\text{Asgn}}{\texttt{C2} : \begin{pmatrix}\cdot&\cdot&\cdot\\\cdot&\cdot&\bullet\\\cdot&\bullet&\cdot\end{pmatrix}}\,\text{Cond}}{\texttt{C1;C2} : \begin{pmatrix}\cdot&\bullet&\cdot\\\cdot&\cdot&\bullet\\\cdot&\bullet&\cdot\end{pmatrix}}\,\text{Comp}$$

Derivation Example II

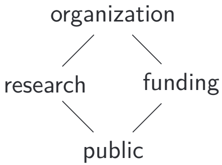
```
while(t[i]!=j){
  s1[i]=j*j;
  s2[i]=1/j;
  i++
}
```

$$\begin{array}{c|ccccc} & \mathtt{t} & \mathtt{i} & \mathtt{j} & \mathtt{s1} & \mathtt{s2} \\ \hline \mathtt{t} & \cdot & 💧 & \cdot & 💧 & 💧 \\ \mathtt{i} & \cdot & \cdot & \cdot & 💧 & 💧 \\ \mathtt{j} & \cdot & 💧 & \cdot & 💧 & 💧 \\ \mathtt{s1} & \cdot & \cdot & \cdot & \cdot & \cdot \\ \mathtt{s2} & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$$

Derivation Example II

```
while(t[i]!=j){
  s1[i]=j*j;
  s2[i]=1/j;
  i++
}
```

$$\begin{array}{c|ccccc} & t & i & j & s1 & s2 \\ \hline t & \cdot & \text{💧} & \cdot & \text{💧} & \text{💧} \\ i & \cdot & \cdot & \cdot & \text{💧} & \text{💧} \\ j & \cdot & \text{💧} & \cdot & \text{💧} & \text{💧} \\ s1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ s2 & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}$$


organization

research

funding

public

TODO: Plan Progress

- ☑ Define programming language and semantics
- ☑ Take an ICC-based data-flow calculus
- ☑ Adjust calculus to track a security property (non-interference)
- ☐ Show useful applications

Practical Advancements and Discoveries

- Prototype implementation to show the ideas is efficient in practice
- Language is extensible to cover functions and OOP
- Adjustable mathematical framework
- Complementary to type system-based analysis

Potential Applications

Idea #1: Taint-analysis or program analysis of noninterference

Given a SFM, security policy, and source and sink variables: find a non-interfering security class assignment if exists, or indicate points of failure.

(this idea has challenges)

Potential Applications

Idea #2: Miscompilation or compiler-introduced issue detection

Map the analysis syntax to high-level and low-level programming language, compare two SFMs to detect issues.

For Java, use a bytecode normalizer[4] to remove bytecode differences.

---

[4] Stefan et al. Schott. "Java Bytecode Normalization for Code Similarity Analysis". In: *38th European Conference on Object-Oriented Programming (ECOOP 2024)*. 2024, 37:1–37:29. DOI: `10.4230/LIPIcs.ECOOP.2024.37`.

Discussion Topics

The Information Flow Calculus – benefits or challenges

- Abstracts the analyzed program
- Static and automatic, no annotations needed etc.
- Flexible: adjustable to increase precision or track other security properties

Utility and potential applications – especially beyond ideas presented so far

- Can target different language syntax and contexts