Relative Secrecy and Semantics of Declassification

Peeter Laud

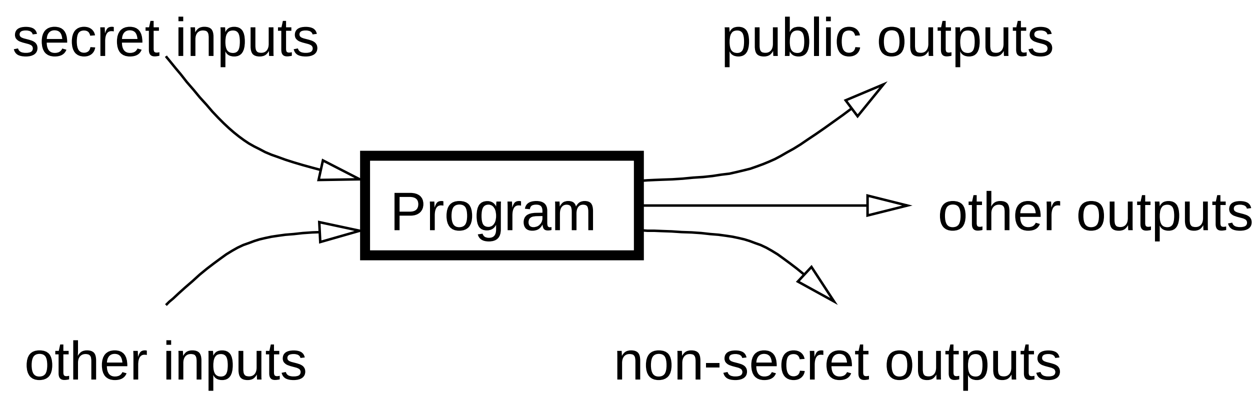
peeter_l@ut.ee

Tartu Ülikool

Cybernetica AS

Supported by the Tiger University Project of Estonian Information Technology Foundation

Problem statement



Does the program satisfy the following secrecy condition?

- The public outputs are made public. . .
- but nothing must be revealed about the secret inputs. . .
- except that we have determined that revealing non-secret outputs will not expose anything sensitive.

Formal definition? Program analysis?

Structure of the talk

- Syntax of language, etc.
- Security definition.
- Program analysis.
  - Analysis domain, simplifying assumptions.
    - These assumptions do not lessen the generality.
  - Transfer functions (a framework for them).
- The $declassify$-statement.
  - Simple program analysis (unconnected to semantics).
  - Rewriting $declassify$-statements.
  - Relation of two analyses.

Program Language

The WHILE-language.

$$\begin{array}{lll} \mathsf{P} & ::= & \mathtt{x} := o(\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}) \\ & | & \mathit{skip} \\ & | & \mathsf{P}_1; \mathsf{P}_2 \\ & | & \mathit{if}\ \mathtt{b}\ \mathit{then}\ \mathsf{P}_1\ \mathit{else}\ \mathsf{P}_2 \\ & | & \mathit{while}\ \mathtt{b}\ \mathit{do}\ \mathsf{P}' \end{array}$$

The set of program states $\mathbf{State}$ is $\mathbf{Var} \to \mathbf{Val}$.
$\mathtt{x}, \mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}, \mathtt{b} \in \mathbf{Var}$, $o \in \mathbf{Op}$.

- Secret inputs — initial values of variables in $\mathbf{Var}_\mathrm{S} \subseteq \mathbf{Var}$
- Public outputs — final values of variables in $\mathbf{Var}_\mathrm{P} \subseteq \mathbf{Var}$
- Non-secret outputs — final values of variables in $\mathbf{Var}_\mathrm{NS}$

Type of deterministic semantics

- The denotational semantics maps program's input state to its output state.

$$[\![\mathsf{P}]\!] : \mathbf{State} \to \mathbf{State}_\perp$$

  - defined inductively over program structure;
  - $\mathbf{State}_\perp = \mathbf{State}\,\dot\cup\{\perp\}$;
  - $\perp$ denotes nontermination.
- For now, our approach is termination-insensitive.
  - Issues with termination are probably orthogonal to other issues.
  - we therefore assume $[\![\mathsf{P}]\!] : \mathbf{State} \to \mathbf{State}$.

Non-interference

Usual definition:
The values of program's public outputs must be determined by the values of its "other" inputs.

$$\exists f : (\mathbf{Var} \backslash \mathbf{Var}_{\mathrm{S}} \to \mathbf{Val}) \to (\mathbf{Var}_{\mathrm{P}} \to \mathbf{Val})$$

such that for all $S \in \mathbf{State}$

$$[\![\mathsf{P}]\!](S)|_{\mathbf{Var}_{\mathrm{P}}} = f(S|_{\mathbf{Var} \backslash \mathbf{Var}_{\mathrm{S}}}) \quad .$$

Relative secrecy

With non-secret outputs:
The values of program's public outputs must be determined by the values of its "other" inputs and its non-secret outputs.

$$\exists f : (\mathbf{Var} \backslash \mathbf{Var}_{\mathrm{S}} \to \mathbf{Val}) \times (\mathbf{Var}_{\mathrm{NS}} \to \mathbf{Val}) \to (\mathbf{Var}_{\mathrm{P}} \to \mathbf{Val})$$

such that for all $S \in \mathbf{State}$

$$[\![\mathsf{P}]\!](S)|_{\mathbf{Var}_{\mathrm{P}}} = f(S|_{\mathbf{Var} \backslash \mathbf{Var}_{\mathrm{S}}}, [\![P]\!](S)|_{\mathbf{Var}_{\mathrm{NS}}}) \quad .$$

We let $S_1 =_X S_2$ denote $S_1|_X = S_2|_X$, where $X \subseteq \mathbf{Var}$.

Relative secrecy

In other words: for all $S_1, S_2 \in \mathbf{State}$:

$$S_1 =_{\mathbf{Var} \setminus \mathbf{Var}_{\mathrm{S}}} S_2 \wedge [\![\mathsf{P}]\!](S_1) =_{\mathbf{Var}_{\mathrm{NS}}} [\![\mathsf{P}]\!](S_2) \Rightarrow$$
$$[\![\mathsf{P}]\!](S_1) =_{\mathbf{Var}_{\mathrm{P}}} [\![\mathsf{P}]\!](S_2)$$

If we assume P does not change the variables in $\mathbf{Var} \setminus \mathbf{Var}_{\mathrm{S}}$ (this assumption is w.l.o.g), then

$$[\![\mathsf{P}]\!](S_1) =_{\mathbf{Var} \setminus \mathbf{Var}_{\mathrm{S}}} [\![\mathsf{P}]\!](S_2) \wedge [\![\mathsf{P}]\!](S_1) =_{\mathbf{Var}_{\mathrm{NS}}} [\![\mathsf{P}]\!](S_2) \Rightarrow$$
$$[\![\mathsf{P}]\!](S_1) =_{\mathbf{Var}_{\mathrm{P}}} [\![\mathsf{P}]\!](S_2)$$

Abstract domain

Given $\mathcal{S} \subseteq \mathbf{State}$ and $X, Y, Z \subseteq \mathbf{Var}$, we are interested if

$$S_1 =_X S_2 \wedge S_1 =_Y S_2 \Rightarrow S_1 =_Z S_2$$

holds for all $S_1, S_2 \in \mathcal{S}$.

It can be found if we know for all $X \subseteq \mathbf{Var}$ and $\mathbf{z} \in \mathbf{Var}$ if

$$S_1 =_X S_2 \Rightarrow S_1(\mathbf{z}) = S_2(\mathbf{z})$$

holds for all $S_1, S_2 \in \mathcal{S}$.

We abstract $\mathcal{P}(\mathbf{State})$ by $\mathcal{P}(\mathcal{P}(\mathbf{Var}) \times \mathbf{Var})$.
Let α be the abstraction function.

Analysis — overall approach

- Let $A_\circ$ be the abstraction of the set of possible input states.
- Apply the abstract semantics of P to $A_\circ$, giving $A_\bullet$.
  - $A_\bullet$ approximates the abstraction of the set of possible output states.
  - It is a conservative approximation — some pairs $(X, \mathtt{z})$ may be missing.
- If $\big((\mathbf{Var} \backslash \mathbf{Var}_{\mathrm{S}}) \cup \mathbf{Var}_{\mathrm{NS}}, \mathtt{x}\big) \in A_\bullet$ for all $\mathtt{x} \in \mathbf{Var}_{\mathrm{P}}$, then consider the program secure.

Properties of abstraction

- Let $A = \alpha(\mathcal{S})$ for some $\mathcal{S} \in \mathbf{State}$. Then
  - $(\{\mathtt{z}\}, \mathtt{z}) \in A$,
  - $(X, \mathtt{z}) \in A \Rightarrow (X \cup Y, \mathtt{z}) \in A$,
  - $(X \cup \{\mathtt{y}\}, \mathtt{z}) \in A \wedge (X, \mathtt{y}) \in A \Rightarrow (X, \mathtt{z}) \in A$

  hold for all $X, Y \subseteq \mathbf{Var}$ and $\mathtt{y}, \mathtt{z} \in \mathbf{Var}$.
- If $A \subseteq \mathcal{P}(\mathbf{Var}) \times \mathbf{Var}$ satisfies these implications then we call A closed.
- The closure of A is the smallest closed set containing A.

Analysis of assignments

- The analysis $\mathcal{A}(\mathtt{x} := o(\mathtt{x_1}, \ldots, \mathtt{x_k}))$, applied to $A_\circ$, will construct $A_\bullet$ by
  - kill $\mathtt{x}$, i.e. remove all $(X, \mathtt{z})$ where $\mathtt{x} \in X$ or $\mathtt{x} = \mathtt{z}$;
  - add $(\{\mathtt{x_1}, \ldots, \mathtt{x_k}\}, \mathtt{x})$;
  - construct the closure.

  (we assume that $\mathtt{x} \notin \{\mathtt{x_1}, \ldots, \mathtt{x_k}\}$)
- If some $\mathtt{x_i}$ can be found from some set $X \subseteq \{\mathtt{x}, \mathtt{x_1}, \ldots, \mathtt{x_k}\}$ after the operation, then also add $(X, \mathtt{x_i})$ during the second step.
  - Example: $\mathtt{y}$ can be found from $\{\mathtt{x}, \mathtt{z}\}$ after $\mathtt{x} := \mathtt{y} + \mathtt{z}$.

Analysis of $skip$ and composition

- $\mathcal{A}(skip)$ is the identity function.
- $\mathcal{A}(\mathsf{P}_1; \mathsf{P}_2) = \mathcal{A}(\mathsf{P}_2) \circ \mathcal{A}(\mathsf{P}_1)$.

Analysis of $if - then - else$

Consider the program $\textit{if}\ \mathtt{b}\ \textit{then}\ \mathsf{P}_1\ \textit{else}\ \mathsf{P}_2$.

- Let $\{\mathtt{x}_1, \ldots, \mathtt{x}_k\} = \mathbf{Var}_{\mathrm{asgn}} \subseteq \mathbf{Var}$ be the set of variables assigned to in P_1 and/or P_2.
- Let $\mathbf{Var}' = \mathbf{Var} \,\dot{\cup}\, \{N, \mathtt{x}_1^{\mathsf{true}}, \ldots, \mathtt{x}_k^{\mathsf{true}}, \mathtt{x}_1^{\mathsf{false}}, \ldots, \mathtt{x}_k^{\mathsf{false}}\}$
- Program at right has the same functionality.
- $\mathsf{P}_1^{\mathsf{true}}$ is P_1, where each $\mathtt{x}_i$ is replaced with $\mathtt{x}_i^{\mathsf{true}}$.
- Similarly for $\mathsf{P}_2^{\mathsf{false}}$.

$\mathtt{N} := \mathtt{b}$
$\mathtt{x}_1^{\mathsf{true}} := \mathtt{x}_1$
$\mathtt{x}_1^{\mathsf{false}} := \mathtt{x}_1$
$\ldots$
$\mathtt{x}_k^{\mathsf{true}} := \mathtt{x}_k$
$\mathtt{x}_k^{\mathsf{false}} := \mathtt{x}_k$
$\mathsf{P}_1^{\mathsf{true}}$
$\mathsf{P}_2^{\mathsf{false}}$
$\mathtt{x}_1 := \mathtt{N}\,?\,\mathtt{x}_1^{\mathsf{true}} : \mathtt{x}_1^{\mathsf{false}}$
$\ldots$
$\mathtt{x}_k := \mathtt{N}\,?\,\mathtt{x}_k^{\mathsf{true}} : \mathtt{x}_k^{\mathsf{false}}$

Analyse the program at right instead.

Analysis of *while*

$\mathcal{A}(\mathit{while}\ \mathtt{b}\ \mathit{do}\ \mathsf{P})$, applied to $A_\circ$, repeatedly applies $\mathcal{A}(\mathit{if}\ \mathtt{b}\ \mathit{then}\ \mathsf{P}\ \mathit{else}\ \mathit{skip})$ to it, until reaching a fix-point.

Correctness follows from

$$[\![\mathit{while}\ \mathtt{b}\ \mathit{do}\ \mathsf{P}]\!] = [\![\mathit{while}\ \mathtt{b}\ \mathit{do}\ \mathsf{P}]\!] \circ [\![\mathit{if}\ \mathtt{b}\ \mathit{then}\ \mathsf{P}\ \mathit{else}\ \mathit{skip}]\!] \ .$$

The declassification statement

- We add the statement

$$declassify(\mathtt{x}),$$

  where $\mathtt{x} \in \mathbf{Var}$ to the language.
- Its semantics is equal to that of $skip$.
- Its intuitive meaning — currently the value of variable $\mathtt{x}$ does not give away anything about the secret inputs.
- This intuitive meaning is reflected in the analysis.

A simple analysis with declassification

- Consider a simple analysis that maps initial public variables to final public variables.
- $\mathcal{B}(\mathsf{P}) : \mathcal{P}(\mathbf{Var}) \rightarrow \mathcal{P}(\mathbf{Var})$, where the domain and range are sets of public variables.

$$\mathcal{B}(\mathtt{x} := o(\mathtt{x_1}, \ldots, \mathtt{x_k}))(B_\circ) = \begin{cases} B_\circ \cup \{\mathtt{x}\}, & \text{if } \mathtt{x_1}, \ldots, \mathtt{x_k} \in B_\circ \\ B_\circ \backslash \{\mathtt{x}\}, & \text{otherwise.} \end{cases}$$

$$\mathcal{B}(\mathit{declassify}(\mathtt{x}))(B_\circ) = B_\circ \cup \{\mathtt{x}\}$$

Other statements: as before.

The relationship of analyses

- Let $B \subseteq \mathbf{Var}$. Given $\mathbf{Var}_{\mathrm{S}}$ and $\mathbf{Var}_{\mathrm{NS}}$, let

  $$\xi(B) := \{\, \big((\mathbf{Var} \backslash \mathbf{Var}_{\mathrm{S}}) \cup \mathbf{Var}_{\mathrm{NS}}, \mathtt{x}\big) : \mathtt{x} \in B\} \ .$$

  The function ξ binds the domains of $\mathcal{B}$ and $\mathcal{A}$.
- We want to define a program transformation $\bar{\cdot}$ and a set $\mathbf{Var}_{\mathrm{NS}}$, such that for all programs P and $B_\circ \subseteq \mathbf{Var}$:

  $$\xi\big(\mathcal{B}(\mathsf{P})(B_\circ)\big) \subseteq \mathcal{A}(\overline{\mathsf{P}})(\xi(B_\circ)) \ .$$

Program transformation (1/3)

Let $\mathtt{d}$ be a new variable. Then

$$\overline{\mathsf{P}} := \left[\mathtt{d} := \mathsf{Nil}; \mathcal{T}(\mathsf{P}, \mathtt{d})\right]$$

and $\mathbf{Var}_{\mathrm{NS}} = \{\mathtt{d}\}$. Here $\mathcal{T}$ works as follows:

- $\mathcal{T}(\mathtt{x} := o(\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}), \mathtt{d}) = \left[\mathtt{x} := o(\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k})\right]$;
- $\mathcal{T}(\mathit{declassify}(\mathtt{x}), \mathtt{d}) := \left[\mathtt{tmp} := \mathtt{d}; \mathtt{d} := (\mathtt{x}, \mathtt{tmp})\right]$, where $\mathtt{tmp}$ is a new variable;
  - Note that in the analysis $\mathcal{A}$, both $\mathtt{y}$ and $\mathtt{z}$ can be found from $\mathtt{x}$ after $\mathtt{x} := (\mathtt{y}, \mathtt{z})$.
- $\mathcal{T}(\mathit{skip}, \mathtt{d}) = \mathit{skip}$;
- $\mathcal{T}(\mathsf{P}_1; \mathsf{P}_2, \mathtt{d}) = \mathcal{T}(\mathsf{P}_1, \mathtt{d}); \mathcal{T}(\mathsf{P}_2, \mathtt{d})$;

Program transformation (2/3)

$$\mathcal{T}(\mathit{if}\ \ b\ \mathit{then}\ \mathsf{P}_1\ \ \mathit{else}\ \mathsf{P}_2, \mathtt{d}) :=$$

$$\mathtt{d}' := \mathsf{Nil}; \left[\mathit{if}\ b\ \mathit{then}\ \mathcal{T}(\mathsf{P}_1, \mathtt{d}')\ \mathit{else}\ \mathcal{T}(\mathsf{P}_2, \mathtt{d}')\right]; \mathtt{tmp} := \mathtt{d}; \mathtt{d} := (\mathtt{d}', \mathtt{tmp})$$

where $\mathtt{d}'$ and $\mathtt{tmp}$ are new variables.

When proving $\xi\big(\mathcal{B}(\mathsf{P})(B_\circ)\big) \subseteq \mathcal{A}(\overline{\mathsf{P}})(\xi(B_\circ))$ for $\mathsf{P} \equiv \mathit{if}\ \ b\ \mathit{then}\ \mathsf{P}_1\ \ \mathit{else}\ \mathsf{P}_2$ by induction over program structure, then the set $\mathbf{Var}_{\mathrm{NS}}$ for P_1 and P_2 additionally contains $\mathtt{d}'$.

Program transformation (3/3)

- To define $\mathcal{T}(\mathit{while}\ b\ \mathit{do}\ \mathsf{P}, \mathtt{d})$, introduce the construct $\cdot^*$ to the programming language.
- The semantics of P^* is the fix-point of iterating $[\![\mathsf{P}]\!]$. Similarly, $\mathcal{A}(\mathsf{P}^*)$ is the fix-point of iterating $\mathcal{A}(\mathsf{P})$.
- $\mathcal{T}(\mathit{while}\ b\ \mathit{do}\ \mathsf{P}, \mathtt{d})$ is defined as

$$\Big[\mathtt{d}' := \mathsf{Nil}; \big[\mathit{if}\ b\ \mathit{then}\ \mathcal{T}(\mathsf{P}, \mathtt{d}')\ \mathit{else}\ \mathit{skip}\big]; \mathtt{tmp} := \mathtt{d}; \mathtt{d} := (\mathtt{d}', \mathtt{tmp})\Big]^*,$$

where $\mathtt{d}'$ and $\mathtt{tmp}$ are new variables.

Addendum to the analysis $\mathcal{A}$

Let the program P be

$$\mathtt{x}_1 := \mathtt{N} \,?\, \mathtt{x}_1^{\mathsf{true}} : \mathtt{x}_1^{\mathsf{false}}; \mathtt{x}_2 := \mathtt{N} \,?\, \mathtt{x}_2^{\mathsf{true}} : \mathtt{x}_2^{\mathsf{false}}; \cdots ; \mathtt{x}_\mathtt{k} := \mathtt{N} \,?\, \mathtt{x}_\mathtt{k}^{\mathsf{true}} : \mathtt{x}_\mathtt{k}^{\mathsf{false}}$$

Let $A_\circ$ be the initial analysis information. Let

$$\begin{array}{lr} X \subseteq \mathbf{Var} \backslash \{\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}\} & (X, \mathtt{N}) \in A_\circ \\ Y \subseteq \{\mathtt{x}_1, \ldots, \mathtt{x}_\mathtt{k}\} & (X \cup Y^{\mathsf{true}}, \mathtt{x}_\mathtt{i}^{\mathsf{true}}) \in A_\circ \\ \mathtt{i} \in \{1, \ldots, \mathtt{k}\} & (X \cup Y^{\mathsf{false}}, \mathtt{x}_\mathtt{i}^{\mathsf{false}}) \in A_\circ \end{array}$$

then we may take $(X \cup Y, \mathtt{x}_\mathtt{i}) \in A_\bullet$.

This addendum is necessary for relating $\mathcal{A}$ and $\mathcal{B}$.

Concluding remarks

- Relative secrecy can be used to give semantics to some constructs.
- It may also be a tool for modularizing the security analysis.
  - Particularly in the case, when the security of different operations has different flavor.
    - Information-theoretic, complexity-theoretic, etc.
- The “right way” of defining the transfer functions is not yet so clear.
  - I.e. the way that gives the most intuitive analysis results.
  - The intuition itself does not yet exist.