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.

\[
\begin{align*}
P & ::= x := o(x_1, \ldots, x_k) \\
    & | \text{skip} \\
    & | P_1; P_2 \\
    & | \text{if } b \text{ then } P_1 \text{ else } P_2 \\
    & | \text{while } b \text{ do } P'
\end{align*}
\]

The set of program states \(\text{State}\) is \(\text{Var} \rightarrow \text{Val}\).
\(x, x_1, \ldots, x_k, b \in \text{Var}, o \in \text{Op}\).

- Secret inputs — initial values of variables in \(\text{Var}_S \subseteq \text{Var}\)
- Public outputs — final values of variables in \(\text{Var}_P \subseteq \text{Var}\)
- Non-secret outputs — final values of variables in \(\text{Var}_{NS}\)
Type of deterministic semantics

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

$$\mathbb{[P]} : \text{State} \rightarrow \text{State}_\bot$$

- defined inductively over program structure;
- $$\text{State}_\bot = \text{State} \cup \{\bot\}$$;
- $$\bot$$ denotes nontermination.

For now, our approach is termination-insensitive.

- Issues with termination are probably orthogonal to other issues.
- we therefore assume $$\mathbb{[P]} : \text{State} \rightarrow \text{State}.$$
Non-interference

Usual definition:
The values of program’s public outputs must be determined by the values of its “other” inputs.

\[ \exists f : (\text{Var} \setminus \text{Var}_S \rightarrow \text{Val}) \rightarrow (\text{Var}_P \rightarrow \text{Val}) \]

such that for all \( S \in \text{State} \)

\[ \llbracket P \rrbracket (S)|_{\text{Var}_P} = f(S|_{\text{Var} \setminus \text{Var}_S}) \cdot \]
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.

\[ \exists f : (\text{Var} \setminus \text{Var}_S \to \text{Val}) \times (\text{Var}_{NS} \to \text{Val}) \to (\text{Var}_P \to \text{Val}) \]

such that for all \( S \in \text{State} \)

\[ [P](S)|_{\text{Var}_P} = f(S|_{\text{Var} \setminus \text{Var}_S}, [P](S)|_{\text{Var}_{NS}}) \cdot \]

We let \( S_1 =_X S_2 \) denote \( S_1|_X = S_2|_X \), where \( X \subseteq \text{Var} \).
Relative secrecy

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

$$S_1 = \text{Var}\backslash\text{Var}_S S_2 \land [P](S_1) = \text{Var}_N S [P](S_2) \Rightarrow$$

$$[P](S_1) = \text{Var}_P [P](S_2)$$

If we assume $P$ does not change the variables in $\text{Var}\backslash\text{Var}_S$ (this assumption is w.l.o.g), then

$$[P](S_1) = \text{Var}\backslash\text{Var}_S [P](S_2) \land [P](S_1) = \text{Var}_N S [P](S_2) \Rightarrow$$

$$[P](S_1) = \text{Var}_P [P](S_2)$$
Abstract domain

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

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

holds for all $S_1, S_2 \in S$.

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

$$S_1 =_X S_2 \Rightarrow S_1(z) = S_2(z)$$

holds for all $S_1, S_2 \in S$.

We abstract $\mathcal{P}(\text{State})$ by $\mathcal{P}(\mathcal{P}(\text{Var}) \times \text{Var})$. Let $\alpha$ 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 $(X, z)$ may be missing.

- If $((\text{Var} \setminus \text{Var}_S) \cup \text{Var}_{NS}, x) \in A_\bullet$ for all $x \in \text{Var}_P$, then consider the program secure.
Properties of abstraction

Let \( A = \alpha(S) \) for some \( S \in \text{State} \). Then

\[
\begin{align*}
(\{z\}, z) & \in A, \\
(X, z) & \in A \Rightarrow (X \cup Y, z) \in A, \\
(X \cup \{y\}, z) & \in A \land (X, y) \in A \Rightarrow (X, z) \in A
\end{align*}
\]

hold for all \( X, Y \subseteq \text{Var} \) and \( y, z \in \text{Var} \).

If \( A \subseteq \mathcal{P}(\text{Var}) \times \text{Var} \) satisfies these implications then we call \( A \) closed.

The closure of \( A \) is the smallest closed set containing \( A \).
Analysis of assignments

The analysis $A(x := o(x_1, \ldots, x_k))$, applied to $A_o$, will construct $A_\bullet$ by:

- kill $x$, i.e. remove all $(X, z)$ where $x \in X$ or $x = z$;
- add $(\{x_1, \ldots, x_k\}, x)$;
- construct the closure.

(we assume that $x \notin \{x_1, \ldots, x_k\}$)

If some $x_i$ can be found from some set $X \subseteq \{x, x_1, \ldots, x_k\}$ after the operation, then also add $(X, x_i)$ during the second step.

Example: $y$ can be found from $\{x, z\}$ after $x := y + z$. 
Analysis of \textit{skip} and composition

- $\mathcal{A}(\text{skip})$ is the identity function.
- $\mathcal{A}(P_1; P_2) = \mathcal{A}(P_2) \circ \mathcal{A}(P_1)$. 
Analysis of if – then – else

Consider the program \( \text{if } b \text{ then } P_1 \text{ else } P_2. \)

- Let \( \{x_1, \ldots, x_k\} = \text{Var}_{\text{asgn}} \subseteq \text{Var} \) be the set of variables assigned to in \( P_1 \) and/or \( P_2. \)

- Let \( \text{Var}' = \text{Var} \cup \{N, x_1^{\text{true}}, \ldots, x_k^{\text{true}}, x_1^{\text{false}}, \ldots, x_k^{\text{false}}\} \)

Program at right has the same functionality.

- \( P_1^{\text{true}} \) is \( P_1 \), where each \( x_i \) is replaced with \( x_i^{\text{true}}. \)

- Similarly for \( P_2^{\text{false}}. \)

Analyse the program at right instead.

\[
\begin{align*}
N &:= b \\
x_1^{\text{true}} &:= x_1 \\
x_1^{\text{false}} &:= x_1 \\
&\cdots \\
x_k^{\text{true}} &:= x_k \\
x_k^{\text{false}} &:= x_k \\
P_1^{\text{true}} &:= \text{true} \quad P_1^{\text{false}} \\
P_2^{\text{true}} &:= \text{false} \\
x_1 &:= N ? x_1^{\text{true}} : x_1^{\text{false}} \\
&\cdots \\
x_k &:= N ? x_k^{\text{true}} : x_k^{\text{false}}
\end{align*}
\]
Analysis of \textit{while}

$A(\text{while } b \text{ do } P)$, applied to $A_0$, repeatedly applies $A(\text{if } b \text{ then } P \text{ else skip})$ to it, until reaching a fix-point.

Correctness follows from

$$\left[\text{while } b \text{ do } P\right] = \left[\text{while } b \text{ do } P\right] \circ \left[\text{if } b \text{ then } P \text{ else skip}\right].$$
The declassification statement

- We add the statement

\[ declassify(x), \]

where \( x \in \text{Var} \) to the language.

- Its semantics is equal to that of \( \text{skip} \).

- Its intuitive meaning — currently the value of variable \( x \) 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.

\[ \mathcal{B}(P) : \mathcal{P}(\text{Var}) \rightarrow \mathcal{P}(\text{Var}) \], where the domain and range are sets of public variables.

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

\[
\mathcal{B}(\text{declassify}(x))(B_\circ) = B_\circ \cup \{x\}
\]

Other statements: as before.
The relationship of analyses

Let \( B \subseteq \text{Var} \). Given \( \text{Var}_S \) and \( \text{Var}_{NS} \), let

\[
\xi(B) := \{ ((\text{Var} \setminus \text{Var}_S) \cup \text{Var}_{NS}, x) : x \in B \} .
\]

The function \( \xi \) binds the domains of \( B \) and \( A \).

We want to define a program transformation \( \vdash \) and a set \( \text{Var}_{NS} \), such that for all programs \( P \) and \( B_\circ \subseteq \text{Var} \):

\[
\xi(\mathcal{B}(P)(B_\circ)) \subseteq A(\overline{P})(\xi(B_\circ)) .
\]
Let $d$ be a new variable. Then

$$\overline{P} := [d := \text{Nil}; \mathcal{T}(P, d)]$$

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

1. $\mathcal{T}(x := o(x_1, \ldots, x_k), d) = [x := o(x_1, \ldots, x_k)]$;
2. $\mathcal{T}(\text{declassify}(x), d) := [\text{tmp} := d; d := (x, \text{tmp})]$, where $\text{tmp}$ is a new variable;
3. Note that in the analysis $\mathcal{A}$, both $y$ and $z$ can be found from $x$ after $x := (y, z)$.
4. $\mathcal{T}(\text{skip}, d) = \text{skip}$;
5. $\mathcal{T}(P_1; P_2, d) = \mathcal{T}(P_1, d); \mathcal{T}(P_2, d)$;
Program transformation (2/3)

\[ \mathcal{T}(\text{if } b \text{ then } P_1 \text{ else } P_2, d) := \]
\[ d' := \text{Nil}; [\text{if } b \text{ then } \mathcal{T}(P_1, d') \text{ else } \mathcal{T}(P_2, d')] ; \text{tmp} := d; d := (d', \text{tmp}) \]

where \(d'\) and \(\text{tmp}\) are new variables.

When proving \(\xi(\mathcal{B}(P)(B_\circ)) \subseteq \mathcal{A}(\overline{P})(\xi(B_\circ))\) for \(P \equiv \text{if } b \text{ then } P_1 \text{ else } P_2\) by induction over program structure, then the set \(\text{Var}_{\text{NS}}\) for \(P_1\) and \(P_2\) additionally contains \(d'\).
To define \( T(\text{while } b \text{ do } P, a) \), introduce the construct \( \cdot^* \) to the programming language.

The semantics of \( P^* \) is the fix-point of iterating \( \llbracket P \rrbracket \). Similarly, \( A(P^*) \) is the fix-point of iterating \( A(P) \).

\( T(\text{while } b \text{ do } P, a) \) is defined as

\[
\left[ d' := \text{Nil}; \ [\text{if } b \text{ then } T(P, d') \text{ else skip}] ; \text{tmp := d}; \ d := (d', \text{tmp}) \right]^*,
\]

where \( d' \) and \( \text{tmp} \) are new variables.
Addendum to the analysis $\mathcal{A}$

Let the program $P$ be

$$x_1 := N \ ? \ x_1^\text{true} : x_1^\text{false}; \ x_2 := N \ ? \ x_2^\text{true} : x_2^\text{false}; \ldots; \ x_k := N \ ? \ x_k^\text{true} : x_k^\text{false}$$

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

$$X \subseteq \text{Var} \setminus \{x_1, \ldots, x_k\} \quad (X, N) \in A_\circ$$

$$Y \subseteq \{x_1, \ldots, x_k\} \quad (X \cup Y^{\text{true}}, x_i^{\text{true}}) \in A_\circ$$

$$i \in \{1, \ldots, k\} \quad (X \cup Y^{\text{false}}, x_i^{\text{false}}) \in A_\circ$$

then we may take $(X \cup Y, x_i) \in A_\bullet$.

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.