Dynamic Information Flow Control

CS 5430

April 16, 2018

A dynamic mechanism checks and/or deduces labels on variables during execution. If an assignment $x := e$ is executed, and if Γ is fixed, then the mechanism checks whether relation $\Gamma(e) \sqcup ctx \sqsubseteq \Gamma(x)$ holds and halts the execution when the check fails. If an assignment $x := e$ is executed, and if Γ is flow sensitive, then the mechanism deduces label $\Gamma(x)$ to be $\Gamma(e) \sqcup ctx$. When execution enters a conditional command, the mechanism augments ctx with the label of the guard.

Consider the if-statement below:

$$\textbf{if } 1 = 1 \textbf{ then } y := 1 \textbf{ else } y := x \textbf{ end} \tag{1}$$

Under a dynamic mechanism, statement (1) would be accepted. This is because, at any execution, only assignment $y := 1$ is executed, and this assignment does not cause any illegal explicit or implicit flows. So, statement (1), which is rejected by the static type system, would be accepted by a dynamic mechanism.

However, dynamic mechanisms are notorious for introducing leakage of information through the checks they perform and the labels they deduce during execution. The example below demonstrates how sensitive information may be leaked through deduced labels:

$$\begin{array}{l} x := 0; \\ \textbf{if } h > 0 \textbf{ then } x := 1 \textbf{ else skip end}; \\ y := x \end{array} \tag{2}$$

Assume the dynamic mechanism uses a flow sensitive mapping Γ, which is initialized as: $\Gamma(x) = L$, $\Gamma(y) = L$, and $\Gamma(h) = H$. Assume also that the final value of y is the output of the program. If $h > 0$ is *false*, then $\Gamma(x)$ remains L, and thus, $\Gamma(y)$ becomes L, at termination. So, the value assigned to y will be a public output. If $h > 0$ is *true*, then $\Gamma(x)$ becomes H, and thus, $\Gamma(y)$ becomes H, at termination. So, no public output will be generated. Thus, the value of $h > 0$ is leaked to public outputs.

Notice that in example (2) the value of $h > 0$ always flows to the value of x. This is because the final value of x is 1 when $h > 0$ is *true*, and 0 when $h > 0$ is *false*. So, at termination, x should always be tagged with H. However, the dynamic mechanism of the example above failed to tag x with H when $h > 0$

was *false*, because during execution no assignment to x was performed in the context of $h > 0$. So, that dynamic mechanism missed to capture the indirect flow from h to x, when $h > 0$ was *false*, because x did not appeared as a target variable[1] in the the taken branch; x was a target variable in the untaken branch.

One way to capture indirect flows to target variables of untaken branches, is to perform an *on-the-fly* static analysis of untaken branches. When the execution (and the analysis) of the taken branch of a conditional statement finishes, the mechanism can analyze the untaken branch, without executing it, to ensure all implicit flows from the context to target variables are captured. Considering again example (2). When $h > 0$ is *false*, and after the taken branch (i.e., **skip**) is executed, an on-the-fly static analysis can be applied to the untaken branch $x := 1$. Specifically, $\Gamma(x)$ would be deduced to be $\Gamma(1) \sqcup \Gamma(h)$, which is label H. So, for every execution, x is tagged with H. Subsequently, for every execution, y is tagged with H, and thus there will always be no public output. Thus, $h > 0$ is no longer leaked to public outputs.

A dynamic enforcement mechanism may leak sensitive information when it decides to halt an execution due to a failed label check. Consider the program below:

$$
\begin{array}{l}
p := 0; \\
\textbf{if } s > 0 \textbf{ then } p := 1 \textbf{ else } s := 1 \textbf{ end}; \\
p := 2
\end{array}
$$

where fixed mapping Γ is: $\Gamma(p) = L$ and $\Gamma(s) = H$. Assume that the final value of p is the (public) output of the program. If $s > 0$ is *true*, then execution is halted to prevent s from implicitly flowing to p through assignment $p := 1$. So, no public output is generated. If $s > 0$ is *false*, then execution terminates normally, and thus one public output is generated. Here, a public output is generated depending on whether the execution is halted or not, which in turn is a decision that depends on $s > 0$. Consequently, $s > 0$ is leaked to public outputs. Current research is focused on designing dynamic mechanisms that do not leak sensitive information when they decide to halt an execution, without making the mechanism too conservative.

As a conclusion, dynamic mechanisms have both advantages and disadvantages comparing to static mechanisms for information flow control. A dynamic mechanism may be less conservative (fewer false negatives) than a static mechanism, but a dynamic mechanism adds run time overhead. Also, a dynamic mechanism may introduce new covert channels due to label deduction and halting decisions.

[1] For any assignment $x := e$, x is called the target variable.

Exercises

Problem 1:
(i) Give a program that is accepted by both the static type system and the dynamic analysis (with on-the-fly static analysis).
(ii) Give another program that satisfies noninterference, but it is rejected by both the static type system and the dynamic analysis (with on-the-fly static analysis).
(ii) If a program is accepted by the static type system, will this program be necessarily accepted by the dynamic analysis, too?

Problem 2: Consider a dynamic mechanism with on-the-fly static analysis and flow sensitive Γ. What are the confidentiality labels that tag variables in the program below, immediately after the execution of $y := 2$? What are the confidentiality labels that tag variables when the program terminates? Assume Γ is initialized as: $\Gamma(x) = H$, $\Gamma(y) = L$, $\Gamma(z) = H$.

$x := 1;$
$y := 2;$
if $z > 0$ **then** $y := 1$ **else** $x := 2$ **end**

Problem 3: Consider a purely dynamic mechanism that decides to halt the execution before entering conditional commands with sensitive guard expressions (i.e., the label of the guard expression is not $\bot$). Can this mechanism leak sensitive information when deciding to halt an execution?