Lecture Notes on Termination-Sensitive Noninterference

15-316: Software Foundations of Security & Privacy
Frank Pfenning

Lecture 13
October 22, 2024

1 Introduction

Our assumption so far in the semantic definition of information flow has been that an attacker can only compare the values of low-security variables in the poststates.

> We define $\Sigma \models \alpha$ secure (program α satisfies *termination-insensitive noninterference with respect to policy* Σ)
> iff
>
> for all ℓ, ω_1, ω_2, ν_1, and ν_2,
>
> whenever $\Sigma \vdash \omega_1 \approx_\ell \omega_2$,
>
> and $\mathsf{eval}\ \omega_1\ \alpha = \nu_1$,
>
> and $\mathsf{eval}\ \omega_2\ \alpha = \nu_2$
>
> then $\Sigma \vdash \nu_1 \approx_\ell \nu_2$.

This ignores that possibility that an attacker might notice that a program does not terminate. For example, with $(x : \mathsf{H})$ and $\mathsf{L} \sqsubset \mathsf{H}$

$$\textbf{if } x > 5 \textbf{ then } (\textbf{while } \top \textbf{ skip}) \textbf{ else skip}$$

an observer can tell if $x > 5$ based on whether the program terminates. But according to our policy it is secure at level L! In this example, that's easy to see because whenever ν_1 and ν_2 both exist, then $\nu_1 \approx_{\mathsf{L}} \nu_2$ because there aren't even any low-security variables. If α does not terminate (that is, has no poststate) in one or both of the two initial states, then the implication is vacuously true.

The first goal of today's lecture is to sharpen our definition of noninterference so that nontermination is considered observable, and the example above would be rejected. The second goal will be to revise the information flow type system so it is sound with respect to the sharpened definition. For general expositions of today's topic, see Hedin and Sabelfeld [2012], Sabelfeld and Myers [2003], and earlier seminal work by Volpano and Smith [1997].

This lecture is also a preparation for the next lecture where we talk about *timing attacks* where information can leak by observing how long a program takes to execute. These kind of attacks are quite common and real concerns in actual systems [Brumley and Boneh, 2005], so it is important to study them from the programming perspective.

2 Termination-Sensitive Noninterference

Intuitively, we'd like to say that if $\mathsf{eval}\ \omega_1 \alpha$ and $\mathsf{eval}\ \omega_2\ \alpha$ both return a poststate, then they still must be equivalent to an observer at level ℓ. Moreoever, if one does not terminate then the other one doesn't either. In this latter case, there are no poststates to compare. The traditional definition [Volpano and Smith, 1997] captures this by stating that if $\mathsf{eval}\ \omega_1\ \alpha = \nu_1$ then there must also be an ν_2 such that $\mathsf{eval}\ \omega_2\ \alpha = \nu_2$, and the two poststates must be observably equivalent. This slightly asymmetric definition is correct due to the symmetry of the $\approx_\ell$ relation. If α terminates in prestate ω_1 but does not in ω_2, then we can just swap the two prestates to show that such a program is not secure. We write $\Sigma \models \alpha\ \mathsf{secure}^\infty$ for termination-sensitive noninterference.

> We define $\Sigma \models \alpha\ \mathsf{secure}^\infty$ (program α satisfies *termination-sensitive noninterference with respect to policy* Σ)
> iff
>
> > for all ℓ, ω_1, ω_2, and ν_1,
> >
> > whenever $\Sigma \vdash \omega_1 \approx_\ell \omega_2$,
> >
> > and $\mathsf{eval}\ \omega_1\ \alpha = \nu_1$,
> >
> > then $\mathsf{eval}\ \omega_2\ \alpha = \nu_2$ for some ν_2
> >
> > such that $\Sigma \vdash \nu_1 \approx_\ell \nu_2$.

Let's make sure our example program (let's call it α_0) is not secure under this definition. For this purpose we have to find a counterexample, that is, two states ω_1 and ω_2 that are low-security equivalent such that α_0 terminates in state ω_1 but not in ω_2. Fortunately, that is easy: for

$$\alpha_0 = (\mathbf{if}\ x > 5\ \mathbf{then}\ (\mathbf{while}\ \top\ \mathbf{skip})\ \mathbf{else}\ \mathbf{skip})$$

we can use

$$\begin{array}{ll} \omega_1 = (x \mapsto 0) & \mathsf{eval}\ \omega_1\ \alpha_0 = (x \mapsto 0) \\ \omega_2 = (x \mapsto 7) & \text{there exists no } \nu_2 \text{ with } \mathsf{eval}\ \omega_2\ \alpha_0 = \nu_2 \end{array}$$

3 Sharpening the Information Flow Type System

The information flow type system so far will admit the program α_0 as secure. The idea is now to revisit the rules to determine how we might need to update them to enforce termination-sensitive noninterference. To parallel the definition of noninterference, we write $\Sigma \vdash \alpha$ secure^∞.

The first thing we note is that the security level of expressions and formulas do not change—they remain the least upper bound of the levels of all variables occurring in them. This is due to our decision that expressions and Boolean conditions always terminate (and are always safe).

So we can focus our attention on the program constructs. In the fragment we treat, every command will be safe, that is, we handle SAFETINY.

Assignment. We first show the prior (termination-insensitive) version of the rule.

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e \ \mathsf{secure}} :=\!F$$

Because expressions are not involved with nontermination, this rule remains unchanged! This is not a proof, of course, it is not difficult to update the one from the termination-insensitive case. If $x := e$ terminates in ω_1 then it also will in ω_2, so the two formulations are equivalent in this case.

$$\frac{\Sigma \vdash e : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \ell' \sqsubseteq \Sigma(x)}{\Sigma \vdash x := e \ \mathsf{secure}^\infty} :=\!F^\infty$$

Sequential Composition. Our previous rule can be summarized as saying that termination-insensitive information flow is *compositional*.

$$\frac{\Sigma \vdash \alpha \ \mathsf{secure} \quad \Sigma \vdash \beta \ \mathsf{secure}}{\Sigma \vdash \alpha \ ; \beta \ \mathsf{secure}} ;F$$

Is that still the case, that is, is termination-sensitive information flow also compositional? It is not entirely obvious when α does not terminate, but we conjecture:

$$\frac{\Sigma \vdash \alpha \ \mathsf{secure}^\infty \quad \Sigma \vdash \beta \ \mathsf{secure}^\infty}{\Sigma \vdash \alpha \ ; \beta \ \mathsf{secure}^\infty} ;F^\infty$$

Let's try to prove or refute the soundness of this. So we set up:

$\Sigma \models \alpha \ \mathsf{secure}^\infty$ (1, first premise)
$\Sigma \models \beta \ \mathsf{secure}^\infty$ (2, second premise)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$ (3, assumption)

eval ω_1 $(\alpha\ ;\ \beta) = \nu_1$	(4, ssumption)
...	
eval ω_2 $(\alpha\ ;\ \beta) = \nu_2$ for some ν_2	(to show)
with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$	(to show)

We start by expanding the definition of evaluation for sequential composition. We show the new lines in blue after each step.

$\Sigma \models \alpha$ secure$^\infty$	(1, first premise)
$\Sigma \models \beta$ secure$^\infty$	(2, second premise)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$	(3, assumption)
eval ω_1 $(\alpha\ ;\ \beta) = \nu_1$	(4, assumption)
eval ω_1 $\alpha = \mu_1$ for some μ_1	(5 and)
and eval μ_1 $\beta = \nu_1$	(6, from (4) by defn. of eval)
...	
eval ω_2 $(\alpha\ ;\ \beta) = \nu_2$ for some ν_2	(to show)
with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$	(to show)

At this point we can use the assumption (coming from the first premise) that α is secure.

$\Sigma \models \alpha$ secure$^\infty$	(1, first premise)
$\Sigma \models \beta$ secure$^\infty$	(2, second premise)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$	(3, assumption)
eval ω_1 $(\alpha\ ;\ \beta) = \nu_1$	(4, assumption)
eval ω_1 $\alpha = \mu_1$ for some μ_1	(5, and)
and eval μ_1 $\beta = \nu_1$	(6, from (4) by defn. of eval)
eval ω_2 $\alpha = \mu_2$ for some μ_2	(7 and)
with $\Sigma \vdash \mu_1 \approx_\ell \mu_2$	(8, from (1), (3), and (5))
...	
eval ω_2 $(\alpha\ ;\ \beta) = \nu_2$ for some ν_2	(to show)
with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$	(to show)

At this point we have an evaluation of β from a prestate μ_1 and a state μ_2 that is ℓ-equivalent to it. So we can use the security for β (coming from the second premise).

$\Sigma \models \alpha$ secure$^\infty$	(1, first premise)
$\Sigma \models \beta$ secure$^\infty$	(2, second premise)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$	(3, assumption)
eval ω_1 $(\alpha\ ;\ \beta) = \nu_1$	(4, assumption)
eval ω_1 $\alpha = \mu_1$ for some μ_1	(5, and)
and eval μ_1 $\beta = \nu_1$	(6, by defn. of eval)
eval ω_2 $\alpha = \mu_2$ for some μ_2 (7 and)	
with $\Sigma \vdash \mu_1 \approx_\ell \mu_2$	(8, from (1), (3), and (5))
eval μ_2 $\beta = \nu_2$ for some ν_2	(9 and)

with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$ (10, from (2), (6), and (8))
...
eval ω_2 $(\alpha\ ;\ \beta) = \nu_2$ for some ν_2 (to show)
with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$ (to show)

At this point we have evaluations of α and β in corresponding states so we can close the gap simply by composing them. The ν_2 we had to find is (not surprisingly) the poststate of β when evaluated in prestate μ_2.

$\Sigma \models \alpha$ secure$^\infty$ (1, first premise)
$\Sigma \models \beta$ secure$^\infty$ (2, second premise)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$ (3, assumption)
eval ω_1 $(\alpha\ ;\ \beta) = \nu_1$ (4, assumption)
eval ω_1 $\alpha = \mu_1$ for some μ_1 (5, and)
and eval μ_1 $\beta = \nu_1$ (6, by defn. of eval)
eval ω_2 $\alpha = \mu_2$ for some μ_2 (7 and)
with $\Sigma \vdash \mu_1 \approx_\ell \mu_2$ (8, from (1), (3), and (5))
eval μ_2 $\beta = \nu_2$ for some ν_2 (9 and)
with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$ (10, from (2), (6), and (8))
eval ω_2 $(\alpha\ ;\ \beta) = \nu_2$ (by defn. of eval)
with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$ (copied from (10))

Good. Our intuition that even termination-sensitive noninterference is compositional has been confirmed.

Skip. This does nothing, so the rule trivially remains the same.

$$\frac{}{\Sigma \vdash \textbf{skip}\ \mathsf{secure}^\infty}\ \textbf{skip}F^\infty$$

Conditionals. The if-then-else construct somehow doesn't seem to be involved in nontermination, only loops are. We therefore expect that the rule doesn't change.

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha\ \mathsf{secure}^\infty \quad \Sigma' \vdash \beta\ \mathsf{secure}^\infty}{\Sigma \vdash \textbf{if}\ P\ \textbf{then}\ \alpha\ \textbf{else}\ \beta\ \mathsf{secure}^\infty}\ \textbf{if}F^\infty$$

The proof is just like before, in the termination-insensitive case. The fact that we now have to account for the nontermination of α or β changes the details of reasoning, of course, but the possibility of nontermination just comes from the nontermination of each branch. So we omit this case.

Loops. Here, we'd expect the crux of the changes because loops introduce non-termination into the language. First, the earlier rule:

$$\frac{\Sigma \vdash P : \ell \quad \ell' = \Sigma(pc) \sqcup \ell \quad \Sigma' = \Sigma[pc \mapsto \ell'] \quad \Sigma' \vdash \alpha \ \mathsf{secure}}{\Sigma \vdash \mathbf{while}\ P\ \alpha \ \mathsf{secure}}\ \mathbf{while}F$$

We see that there are essentially two reasons why a while loop may leak information using nontermination (with $(x : \mathsf{H})$)

1. In the example $\mathbf{if}\ x > 5\ \mathbf{then}\ (\mathbf{while}\ \top\ \mathbf{skip})\ \mathbf{else}\ \mathbf{skip}$ it is because the loop is in a context where we have $pc : \mathsf{H}$.
2. In the example $\mathbf{while}\ x = 8\ \mathbf{skip}$ it is because the loop guard is of high security.

Since nontermination is observed "at the outermost level" and not by a variable in the local context, it seems we have to require that both the security level of the pc and the guard are $\perp$, that is, the least element of the lattice. This means that $\ell' = \Sigma(pc) \sqcup \ell$ should also be $\perp$, and we don't need to update the security level of pc.

$$\frac{\Sigma \vdash P : \perp \quad \Sigma(pc) = \perp \quad \Sigma \vdash \alpha \ \mathsf{secure}^\infty}{\Sigma \vdash \mathbf{while}\ P\ \alpha \ \mathsf{secure}^\infty}\ \mathbf{while}F^\infty$$

Is this sufficient to guarantee termination-sensitive noninterference? We set up:

$\Sigma \models \alpha\ \mathsf{secure}^\infty$ (1, premise)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$ (2, assumption)
$\mathsf{eval}\ \omega_1\ (\mathbf{while}\ P\ \alpha) = \nu_1$ (3, assumption)
...
$\mathsf{eval}\ \omega_2\ (\mathbf{while}\ P\ \alpha) = \nu_2$ for some ν_2 (to show)
with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$ (to show)

At this point, there is a small hiccup: the evaluation of a loop may refer back again to the evaluation of the same loop. We therefore use an auxiliary induction on the number of times we iterate the loop when computing $\mathsf{eval}\ \omega_1\ (\mathbf{while}\ P\ \alpha)$ to ν_1. We know it does compute a poststate, so the number of steps is finite and the induction makes sense. Let's call that number n (which is also the notation we chose in the semantic definition of $\omega[\![\mathbf{while}\ P\ \alpha]\!]^n\nu$, see page L4.6 of Lecture 4).

Case: $n = 0$. Then $\mathsf{eval}_{\mathbb{B}}\ \omega_1\ P = \perp$ and $\nu_1 = \omega_1$. Because P only contains variables of security level $\perp \sqsubseteq \ell$ and $\Sigma \vdash \omega_1 \approx_\ell \omega_2$, we also have $\mathsf{eval}_{\mathbb{B}}\ \omega_2\ P = \perp$. (See Lemma 2 on page L11.4 of Lecture 11.) Therefore $\nu_2 = \omega_2$ will satisfy the requirements of the theorem.

Case: $n > 0$. Then $\mathsf{eval}_{\mathbb{B}}\ \omega_1\ P = \top$ and $\mathsf{eval}\ \omega_1\ (\alpha\ ;\ \mathbf{while}\ P\ \alpha) = \nu_1$ As in the previous case $\mathsf{eval}_{\mathbb{B}}\ \omega_2\ P = \top$ and it remains to show that $\mathsf{eval}\ \omega_2\ (\alpha\ ;\ \mathbf{while}\ P\ \alpha) = \nu_2$ for some ν_2 with $\Sigma \vdash \nu_1 \approx_\ell \nu_2$.

As in the case for composition (and exploiting the security of α that comes from the premise) this reduces to showing that $\mathsf{eval}\ \mu_1\ (\mathbf{while}\ P\ \alpha) = \nu_1$ implies $\mathsf{eval}\ \mu_2\ (\mathbf{while}\ P\ \alpha) = \nu_2$ for some ν_2 indistinguishable from ν_1 at level ℓ, with the knowledge that $\Sigma \vdash \mu_1 \approx_\ell \mu_2$. This is true because of the induction hypothesis on $n - 1 < n$, which is the remaining number of times the loop is traversed.

Tests. Recall that $\mathbf{test}\ P$ aborts if P is false. This means it represents another type of program (besides a loop) that does not have a poststate. Lumping together aborting a program and nontermination may be questionable in practice, but if we consider "nontermination" simply as representing the absence of a poststate, we would have defined

$$\mathbf{test}\ P \triangleq \mathbf{while}\ \neg P\ \mathbf{skip}$$

Then the rule derived from these considerations would be:

$$\frac{\Sigma \vdash P : \bot \quad \Sigma(pc) = \bot}{\Sigma \vdash \mathbf{test}\ P\ \mathsf{secure}^\infty}\ \mathbf{test}F^\infty$$

All together we obtain soundness.

Theorem 1 (Soundness of termination-sensitive information flow types) *If* $\Sigma \vdash \alpha\ \mathsf{secure}^\infty$ *then* $\Sigma \models \alpha\ \mathsf{secure}^\infty$

Proof: All the rules are *sound*, that is, the preserve semantic validity of the premises. By a trivial induction over the derivation of $\Sigma \vdash \alpha\ \mathsf{secure}^\infty$, every program judged secure satisfies termination-sensitive noninterference. □

4 Further Discussion

The soundness of all rules for $\Sigma \vdash \alpha\ \mathsf{secure}^\infty$ guarantees that it implies $\Sigma \models \alpha\ \mathsf{secure}^\infty$, that is, termination-sensitive information flow. The other direction (completeness) is not true, and there are simple counterexamples.

We also have the intuition that it should be *stricter* than the termination-insensitive one. That should be true syntactically, in the type system, and semantically, with respect to noninterference. Let's check that:

Theorem 2 *If* $\Sigma \models \alpha\ \mathsf{secure}^\infty$ *then* $\Sigma \models \alpha\ \mathsf{secure}$.

Proof: We set up:

$\Sigma \models \alpha\ \mathsf{secure}^\infty$	(1, assumption)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$	(2, assumption)
$\mathsf{eval}\ \omega_1\ \alpha = \nu_1$	(3, assumption)
$\mathsf{eval}\ \omega_2\ \alpha = \nu_2$	(4, assumption)
...	
$\Sigma \vdash \nu_1 \approx_\ell \nu_2$	(to show)

We can now apply the definition of secure$^\infty$.

$\Sigma \models \alpha\ \mathsf{secure}^\infty$	(1, assumption)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$	(2, assumption)
$\mathsf{eval}\ \omega_1\ \alpha = \nu_1$	(3, assumption)
$\mathsf{eval}\ \omega_2\ \alpha = \nu_2$	(4, assumption)
$\mathsf{eval}\ \omega_2\ \alpha = \nu_2'$ for some ν_2' with	(5 and)
$\Sigma \vdash \nu_1 \approx_\ell \nu_2'$	(6, from (1), (2), and (3))
...	
$\Sigma \vdash \nu_1 \approx_\ell \nu_2$	(to show)

Since our language is deterministic, we have $\nu_2 = \nu_2'$ and the proof is complete.

$\Sigma \models \alpha\ \mathsf{secure}^\infty$	(1, assumption)
$\Sigma \vdash \omega_1 \approx_\ell \omega_2$	(2, assumption)
$\mathsf{eval}\ \omega_1\ \alpha = \nu_1$	(3, assumption)
$\mathsf{eval}\ \omega_2\ \alpha = \nu_2$	(4, assumption)
$\mathsf{eval}\ \omega_2\ \alpha = \nu_2'$ for some ν_2' with	(5 and)
$\Sigma \vdash \nu_1 \approx_\ell \nu_2'$	(6, from (1), (2), and (3))
$\Sigma \vdash \nu_1 \approx_\ell \nu_2$	(7, from (4), (5), and (6) by determinism)

□

Any syntactic (decidable) type system for a Turing-complete language may be considered an approximation of a true semantic property. This is the essence of Rice's theorem. Any type system that restricts information flow then represents a tradeoff between the kind of programs that are allowed, and the simplicity and uniformity of the system. This is difficult to discuss in the abstract, so in Lab 2 you will have the opportunity to explore it a bit yourself. The information flow type systems from the last three lectures all seems to represent reasonable compromises: the rules are sound and quite simple, and many programs can be written (even if some require declassification).

Perhaps the most significant issue is that there exist further *side channels* by which information can be obtain, most notably perhaps by observing program run-time. We will complete this investigation in the next lecture.q

References

David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, August 2005.

Daniel Hedin and Andrei Sabelfeld. A perspective on information-flow control. In Tobias Nipkow, Orna Grumberg, and Benedikt Hauptmann, editors, *Software Safety and Security - Tools for Analysis and Verification*, pages 319–347. IOS Press, 2012.

Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *Journal on Selected Areas in Communications*, 21(1):5–19, January 2003.

Dennis M. Volpano and Geoffrey Smith. Eliminating covert flows with minimum typings. In *10th Computer Security Foundations Workshop (CSFW 1997)*, pages 156–168. IEEE Computer Society, June 1997.