Lecture Notes on Information Flow

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

Lecture 11
October 1, 2024

1 Introduction

In the last lecture we informally introduced a notion of *information flow* and a type system to ensure *confidentiality*, meaning that high security data do not flow to low security variables. The rules are summarized in Section 5. They express an information flow policy as a mapping Σ from variables to security levels, arranged in a semilattice, including a ghost variable pc to track *implicit flows* due to conditional tests. We often worked with the two element lattice with levels H (for *high*) and L (for *low*) with $\mathsf{L} \sqsubset \mathsf{H}$.

We assumed that an attacker can see the program and control or see the values of all low security variables before and after computation. We also assumed the attacker can not observe nontermination (including abort after a failed test); some of the rules would have to be changed to account for that and still capture confidentiality.

These rules are *syntactic* and it is straightforward to see that they define an algorithm for deriving that a program α is secure with respect to a security policy Σ that assigns security levels.

What was missing was a formal *semantic* definition of information flow security and a proof that the rules are *sound* with respect to this definition. That's the goal of today's lecture. The material is adapted mostly from Volpano et al. [1996].

2 Noninterference

We have already seen that information flow is not a property of a single trace. Simplifying the example further, consider the security policy $\Sigma_0 = (x : \mathsf{H}, y : \mathsf{L})$, the initial state $x = 0, y = 0$ and the programs

$$y := x + 1 \qquad\qquad y := 1$$

The traces are the same, consisting of just one transition

$$(x = 0, y = 0) \Rightarrow (x = 0, y = 1)$$

The program on the left violates our policy, flowing information from x to y to the extent an attacker can fully recover the initial value of x from the value of y in the final state. The program on the right permits no such inference.

If we have just two security levels (L and H) we'd like to formalize our attacker model using a notion of *observation*. We consider the program secure if two initial states are equivalent as far as the attacker can see, then the final states must also be equivalent as far as the attacker can see. If that were not the case, the attacker may be able to make an inference about the secret (hidden) part of the initial state.

We define when two states ω_1 and ω_2 are *observationally equivalent at level* L *with respect to security policy* Σ, written as $\Sigma \vdash \omega_1 \approx_{\mathsf{L}} \omega_2$:

> We define $\Sigma \vdash \omega_1 \approx_{\mathsf{L}} \omega_2$ iff $\omega_1(x) = \omega_2(x)$ for all x such that $\Sigma(x) = \mathsf{L}$.

In our tiny example (with $\Sigma_0 = (x : \mathsf{H}, y : \mathsf{L})$), we have (among others):

$$\begin{array}{lcl} \Sigma_0 \vdash (x = 0, y = 0) & \approx_{\mathsf{L}} & (x = 0, y = 0) \\ \Sigma_0 \vdash (x = 0, y = 0) & \approx_{\mathsf{L}} & (x = 1, y = 0) \\ \Sigma_0 \vdash (x = 0, y = 1) & \not\approx_{\mathsf{L}} & (x = 0, y = 2) \\ \Sigma_0 \vdash (x = 0, y = 1) & \not\approx_{\mathsf{L}} & (x = 1, y = 2) \end{array}$$

Next we define that a program α satisfies *noninterference* if values of high security variables do not affect the outcome as it may be observed at low security. The outcome here is simply the result of evaluation (as defined in Lecture 9).

> We define that program α satisfies *noninterference with respect to policy* Σ
> iff for all ω_1, ω_2, ν_1, and ν_2,
> $\Sigma \vdash \omega_1 \approx_{\mathsf{L}} \omega_2$, $\mathsf{eval}\ \omega_1\ \alpha = \nu_1$, and $\mathsf{eval}\ \omega_2\ \alpha = \nu_2$ implies $\Sigma \vdash \nu_1 \approx_{\mathsf{L}} \nu_2$.

Since evaluation is deterministic, given ω_1 and ω_2, there will be at most one pair ν_1 and ν_2. Noninterference does not talk about the case where there is no final state, which is why this condition is called *termination-insensitive noninterference*.

Let's apply it to our example. We want to show that the first program, $y := x+1$ does **not** satisfy noninterference. That is, we need to find to states, indistinguishable at level L that has distinguishable outcomes. So we pick

$$\omega_1 = (x = 0, y = 0) \text{ and } \omega_2 = (x = 1, y = 0)$$

As we already determined, we have $\Sigma_0 \vdash \omega_1 \approx_{\mathsf{L}} \omega_2$. We further have

$$\begin{array}{lcl} \mathsf{eval}\ \omega_1\ (y := x + 1) & = & (x = 0, y = 1) \\ \mathsf{eval}\ \omega_2\ (y := x + 1) & = & (x = 1, y = 2) \end{array}$$

and $\Sigma_0 \vdash (x = 0, y = 1) \not\approx_{\mathsf{L}} (x = 1, y = 2)$ because the low-observable outcomes for y are different.

On the other hand, for any two (relevant) states that are low-observably equivalent

$$\omega_1 = (x = a, y = b) \text{ and } \omega_2 = (x = a', y = b)$$

we have

$$\begin{array}{rcl} \mathsf{eval}\ \omega_1\ (y := 1) & = & (x = a, y = 1) \\ \mathsf{eval}\ \omega_2\ (y := 1) & = & (x = a', y = 1) \end{array}$$

and

$$\Sigma_0 \vdash (x = a, y = 1) \approx_{\mathsf{L}} (x = a', y = 1)$$

So the program $y := 1$ *does* satisfy noninterference.

By analogous reasoning,[1] the program $y := (x - x) + 1$ also satisfies noninterference, even though our type system would reject it. This is the first example showing the *incompleteness* of the proposed type system with respect to information flow. Static type systems often have to make *sound* (and decidable) approximations to some underlying semantic property (which is often undecidable).

Before we move on, we generalize the noninterference property to an arbitrary semilattice of security levels. First, observations are restricted to all levels *below* a given security level ℓ:

> We define $\Sigma \vdash \omega_1 \approx_\ell \omega_2$ iff $\omega_1(x) = \omega_2(x)$ for all x such that $\Sigma(x) \sqsubseteq \ell$.

From this, semantic security with respect to an information flow policy Σ generalizes the previous definition in a straightforward way.

> We define $\Sigma \models \alpha$ secure iff for all ω_1, ω_2, ν_1, ν_2, and ℓ
> $\Sigma \vdash \omega_1 \approx_\ell \omega_2$, $\mathsf{eval}\ \omega_1\ \alpha = \nu_1$, and $\mathsf{eval}\ \omega_2\ \alpha = \nu_2$ implies $\Sigma \vdash \nu_1 \approx_\ell \nu_2$.

The soundness property now states:

> If $\Sigma \vdash \alpha$ secure then $\Sigma \models \alpha$ secure

The completeness property would go in the other direction, but it does not hold as the small example $y := (x - x) + 1$ from above shows. We will have another example at the end of this lecture.

3 Read Levels of Expressions and Formulas

We will need some properties of expressions and formulas. Usually, we would wait until we find we need them in proving the main theorem (in this case, soundness), but we show them first because they bring up the important proof technique of *rule induction*.

[1] we did not discover this in lecture

The first is that expressions read only below their security level. The proof is by rule induction, which means we have to show that all rules preserve the stated property: if we assume it for all premises then it must hold for the conclusion. We have already implicitly used it when proving soundness of inference rules (for example, for sequent calculus for propositional logic and dynamic logic). The reason that every sequent we can derive is valid is that each inference rule preserves validity.

Lemma 1 (Expression Read Level) *If* $\Sigma \vdash e : \ell$ *then for every* $x \in \mathsf{use}\ e$, $\Sigma(x) \sqsubseteq \ell$.

Proof: By rule induction on the derivation of $\Sigma \vdash e : \ell$. We consider each case in turn.

Case:

$$\frac{}{\Sigma \vdash c : \bot}\ \mathsf{const}F$$

Then $\mathsf{use}\ c = \{\,\}$ and the property is vacuously true.

Case:

$$\frac{\Sigma(x) = \ell}{\Sigma \vdash x : \ell}\ \mathsf{var}F$$

Then $\mathsf{use}\ x = \{x\}$ and $\Sigma(x) = \ell$, so our property is satisfied by reflexivity ($\Sigma(x) = \ell \sqsubseteq \ell$).

Case:

$$\frac{\Sigma \vdash e_1 : \ell_1 \quad \Sigma \vdash e_2 : \ell_2}{\Sigma \vdash e_1 + e_2 : \ell_1 \sqcup \ell_2}\ +F$$

We have

for all $x \in \mathsf{use}\ e_1$, $\Sigma(x) \sqsubseteq \ell_1$	(by ind. hyp.)
for all $x \in \mathsf{use}\ e_2$, $\Sigma(x) \sqsubseteq \ell_2$	(by ind. hyp.)
$x \in \mathsf{use}\ (e_1 + e_2)$	(assumption)
$x \in \mathsf{use}\ e_1$ or $x \in \mathsf{use}\ e_2$	(by defn. of use)
$\Sigma(x) \sqsubseteq \ell_1$ or $\Sigma(x) \sqsubseteq \ell_2$	(from uses of the ind. hyp.)
$\Sigma(x) \sqsubseteq \ell_1 \sqcup \ell_2$	(by property of $\sqcup$)

□

The next lemma has the same kind of proof, so we won't bother writing it out.

Lemma 2 (Formula Read Level) *If* $\Sigma \vdash P : \ell$ *then for every* $x \in \mathsf{use}\ P$, $\Sigma(x) \sqsubseteq \ell$.

Proof: By rule induction, as in the proof of Lemma 1. □

4 Soundness of the Information Flow Type System

We want to prove that $\Sigma \vdash \alpha$ secure then $\Sigma \models \alpha$ secure. As we might expect from the proofs in the preceding section, this should follow by rule induction on $\Sigma \vdash \alpha$ secure.

Before doing this more rigorously, we examine assignment, one of the base cases.

Theorem 3 (Soundness of Information Flow Types) *If* $\Sigma \vdash \alpha$ secure *then* $\Sigma \models \alpha$ secure

Proof: By rule induction on $\Sigma \vdash \alpha$ secure. We'll need one more lemma, which we state and prove later for pedagogical purposes.

Case:

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

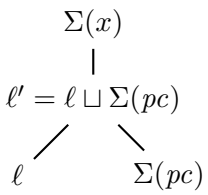
We have to show that $\Sigma \models x := e$ secure. We have the following setup:

$\Sigma \vdash \omega_1 \approx_k \omega_2$	(assumption)
eval ω_1 $(x := e) = \nu_1$	(assumption)
eval ω_2 $(x := e) = \nu_2$	(assumption)
...	
$\Sigma \vdash \nu_1 \approx_k \nu_2$	(to show)

By the rules for evaluation we obtain

$\nu_1 = \omega_1[x \mapsto c_1]$ for $c_1 = \text{eval}_{\mathbb{Z}}\ \omega_1\ e$	(by defn. of eval)
$\nu_2 = \omega_2[x \mapsto c_2]$ for $c_2 = \text{eval}_{\mathbb{Z}}\ \omega_2\ e$	(by defn. of eval)

Let's also depict what we know about the security levels.



At this point we distinguish two cases: $\Sigma(x) \sqsubseteq k$ and $\Sigma(x) \not\sqsubseteq k$.

If $\Sigma(x) \sqsubseteq k$ then also $\ell \sqsubseteq k$ by transitivity. Since $\Sigma \vdash e : \ell$ and $\Sigma \vdash \omega_1 \approx_k \omega_2$ we conclude by Lemma 1 that $c_1 = c_2$ and so $\Sigma \vdash \omega_1[x \mapsto c_1] \approx_k \omega_2[x \mapsto c_2]$.

If $\Sigma(x) \not\sqsubseteq k$ then $\Sigma \vdash \omega_1[x \mapsto c_1] \approx_k \omega_2[x \mapsto c_2]$ because x is not observable at level k and $\Sigma \vdash \omega_1 \approx_k \omega_2$.

Case:

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

We set up this case:

$\Sigma \vdash \omega_1 \approx_k \omega_2$ (assumption)
$\mathsf{eval}\ \omega_1\ (\alpha \,;\, \beta) = \nu_1$ (assumption)
$\mathsf{eval}\ \omega_2\ (\alpha \,;\, \beta) = \nu_2$ (assumption)
...
$\Sigma \vdash \nu_1 \approx_k \nu_2$ (to show)

We can reason with the definition of eval.

$\mathsf{eval}\ \omega_1\ \alpha = \mu_1$ and $\mathsf{eval}\ \mu_1\ \beta = \nu_1$ for some μ_1 (by defn. of eval)
$\mathsf{eval}\ \omega_2\ \alpha = \mu_2$ and $\mathsf{eval}\ \mu_2\ \beta = \nu_2$ for some μ_2 (by defn. of eval)

Next, by induction hypothesis $\Sigma \models \alpha$ secure and together with the assumption about $\Sigma \vdash \omega_1 \approx_k \omega_2$ we get

$\Sigma \vdash \mu_1 \approx_k \mu_2$ (by ind. hyp. on first premise)

This is the fact we need to apply the induction hypothesis on the security of β, because β is evaluated in states μ_1 and μ_2.

$\Sigma \vdash \nu_1 \approx_k \nu_2$ (by ind. hyp. on second premise)

Fortunately, this is just what we needed to show.

Case:

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

This is the most difficult case since it involves implicit flows and therefore the ghost variable pc. It will require a lemma. But first we set up:

$\Sigma \vdash \omega_1 \approx_k \omega_2$ (assumption)
$\mathsf{eval}\ \omega_1\ (\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta) = \nu_1$ (assumption)
$\mathsf{eval}\ \omega_2\ (\mathbf{if}\ P\ \mathbf{then}\ \alpha\ \mathbf{else}\ \beta) = \nu_2$ (assumption)
...
$\Sigma \vdash \nu_1 \approx_k \nu_2$ (to show)

Here is what we know about the security levels in the rule.

$$\begin{array}{ccc} \ell' = \ell \sqcup \Sigma(pc) & & \\ \diagup & \diagdown & \\ \ell & & \Sigma(pc) \end{array}$$

Again, we distinguish two cases: $\ell' \sqsubseteq k$ and $\ell' \not\sqsubseteq k$.

If $\ell' \sqsubseteq k$ then also $l \sqsubseteq k$ and $\mathsf{eval}_{\mathbb{B}}\ \omega_1\ P = \mathsf{eval}_{\mathbb{B}}\ \omega_2\ P = b$ for some Boolean b by Lemma 2. If $b = \top$ we can apply the induction hypothesis to the first premise and the evaluations $\mathsf{eval}\ \omega_1\ \alpha = \nu_1$ and $\mathsf{eval}\ \omega_2\ \alpha = \nu_2$.

If $\ell' \not\sqsubseteq k$ then $\mathsf{eval}_{\mathbb{B}}\ \omega_1\ P$ might be different from $\mathsf{eval}_{\mathbb{B}}\ \omega_2\ P$ and the different branches might be taken. In that case we need to prove that, nevertheless, $\Sigma \vdash \mathsf{eval}\ \omega_1\ \alpha \approx_k \mathsf{eval}\ \omega_2\ \beta$.

At this point we can only conclude that because the security level of the pc is increased to ℓ'. The salient property is that if $\Sigma' \vdash \alpha\ \mathsf{secure}$ and $\Sigma'(pc) = \ell'$, then α *will only write to variables with security level above* ℓ'. And the same for β. (These are instances of Lemma 4.) Fortunately, $\ell' \not\sqsubseteq k$, so none of the writes of α and β will be observable at level k or below and $\Sigma \vdash \nu_1 \approx_k \nu_2$.

Case:

$$\frac{}{\Sigma \vdash \mathbf{test}\ P\ \mathsf{secure}}\ \mathbf{test}F$$

We set up:

$\Sigma \vdash \omega_1 \approx_k \omega_2$ (assumption)
$\mathsf{eval}\ \omega_1\ (\mathbf{test}\ P) = \nu_1$ (assumption)
$\mathsf{eval}\ \omega_2\ (\mathbf{test}\ P) = \nu_2$ (assumption)
...
$\Sigma \vdash \nu_1 \approx_k \nu_2$ (to show)

By definition of eval we know that P must evaluate to true in both ω_1 and ω_2 and $\nu_1 = \omega_1$ and $\nu_2 = \omega_2$. So the desired conclusion follows immediately from the strong assumption that a poststate actually exists.

Case:

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

This case is similar to the case for conditional and we leave it to the reader. Writing it out is a good test of your understanding!

□

What remains is the *confinement lemma* relating the security level of the ghost variable pc to the writing behavior of the program. We define maydef α as the set of variables that a program may assign a value to—just all the left-hand sides of assignments in α. This can include more variables than the set def α which includes only those variables that α must write to.

Lemma 4 (Confinement) *If* $\Sigma \vdash \alpha$ secure *then for every* $x \in$ maydef α, $\Sigma(pc) \sqsubseteq \Sigma(x)$.

Proof: By rule induction on $\Sigma \vdash \alpha$ secure. The key observation is that for an assignment $x := e$ the rules require that $\Sigma(pc) \sqsubseteq \Sigma(x)$ and that the other rules only maintain or raise the level. We elide the case-by-case detail. □

We conclude the lecture with another example that, according to the definition, satisfies noninterference but we cannot derive that within the type system.

$$\textbf{if } x = 0 \textbf{ then } y := 1 \textbf{ else } y := 1$$

In the type system this fails with $\Sigma_0 = (x : \mathsf{H}, y : \mathsf{L})$ because the assignments in the two branches are to low-security variables. It is nevertheless secure because the two values of y are the same, so the observable outcomes at low level are equal.

5 Summary: Information Flow Type System

References

Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

$\boxed{\Sigma \vdash e : \ell}$

$$\frac{\Sigma(x) = \ell}{\Sigma \vdash x : \ell}\ \mathsf{var}F \qquad \frac{}{\Sigma \vdash c : \bot}\ \mathsf{const}F \qquad \frac{\Sigma \vdash e_1 : \ell_1 \quad \Sigma \vdash e_2 : \ell_2 \quad \ell = \ell_1 \sqcup \ell_2}{\Sigma \vdash e_1 + e_2 : \ell}\ +F$$

$\boxed{\Sigma \vdash P : \ell}$

$$\frac{\Sigma \vdash e_1 : \ell_1 \quad \Sigma \vdash e_2 : \ell_2}{\Sigma \vdash e_1 \le e_2 : \ell_1 \sqcup \ell_2}\ \le F \qquad \frac{}{\Sigma \vdash \top : \bot}\ \top F \qquad \frac{\Sigma \vdash P : \ell_1 \quad \Sigma \vdash Q : \ell_2}{\Sigma \vdash P \wedge Q : \ell_1 \sqcup \ell_2}\ \wedge F$$

$\boxed{\Sigma \vdash \alpha\ \mathsf{secure}}$

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

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

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

$$\frac{}{\Sigma \vdash \mathbf{test}\ P\ \mathsf{secure}}\ \mathbf{test}F$$

Figure 1: Information Flow Type System
Termination insensitive, ensuring confidentiality