Language-Based Security

Notes from OPLSS@BU 2024, Lecture 2

Lecturer: Stephen Chong
Scribe: Simon Guo

6/11/24

1 Information Flow

Access control doesn't capture what happens to information after access, which involves information flow. Our current definition of information flow is extensional, which means it is about the external behavior of the system. Cohen (1976) introduced **strong dependency**: consider a (deterministic) system H whose inputs include entity A and whose outputs include entity B. Output B **strongly depends** on input A if there exist two executions of H where the inputs differ only for entity A and the output B differs.

This is essentially the key definition of noninterference today. Then security is the absence of certain strong dependencies.

1.1 Example

Assume two security levels, $High$ and Low. Use the same definition as before:

$$\sigma_1 =_{Low} \sigma_2 \iff \forall x.\, \Gamma(x) = Low \Rightarrow \sigma_1(x) = \sigma_2(x).$$

A program c is **noninterfering** if

$$\forall \sigma_1, \sigma_2, \sigma_1', \sigma_2'.\ \sigma_1 =_{Low} \sigma_2 \wedge \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \Rightarrow \sigma_1' =_{Low} \sigma_2'.$$

This is equivalent to having no strong dependencies from high inputs to low outputs.

1.2 Beyond Noninterference for IMP

We need to specify the entities present and what information flows are allowed between them, including

- The **computation model** - the entities that are manipulated during system executions.

- The **threat model** - the entities with which the adversaries interact.
- **Allowed/forbidden information flow** - the control flows between the system and the adversary.

There has been much research into these specifications to more closely model real-world systems.

1.3 Labels and Flow Relations

Labels are syntactic objects associated with entities of a system. Some examples are

- Sensitivity: $Secret, Public$
- Names: $Alice, Charlie, Bob$
- Government security level and compartment:

  $\{(Level, Compartment) \mid Level \in \{Public, Confidential, Secret, TopSecret\} \wedge Compartment \in \{Nuclear, Cryptography, Biological...\}\}$

Info-flow then describes flows between entities based on labels, though they are not themselves policies. Info-flow policies are often represented as flow relations $\sqsubseteq$ on a set Λ of labels, where if $\ell_1 \sqsubseteq \ell_2$, then info is allowed to flow from ℓ_1 to ℓ_2 This flow relation should have reflexivity and transitivity, making it a pre-order. It can also be made a partial order by adding antisymmetry.

We can also make this a join-semi-lattice flow relation by adding a least-upper-bound $\sqcup$ with the following properties:

- Upper Bound: $\forall \ell_1, \ell_2 \in \Lambda, \ell_1 \sqsubseteq \ell_1 \sqcup \ell_2 \ \wedge \ \ell_2 \sqsubseteq \ell_1 \sqcup \ell_2$.
- Least Upper Bound: $\forall \ell_1, \ell_2, \ell_3 \in \Lambda, \ell_1 \sqsubseteq \ell_3 \wedge \ell_2 \sqsubseteq \ell_3 \Rightarrow \ell_1 \sqcup \ell_2 \sqsubseteq \ell_3$.

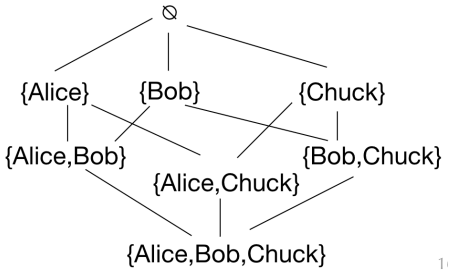
This enables modeling of if-then-else like statements, or combining data from different labels. $\sqcup$ needs to be as precise as possible (hence least upper bound), otherwise the following will be considered a violation

$$c = a \oplus b; d1 = c; d2 = c$$

if multiple upper bounds of ℓ_a and ℓ_b are used for ℓ_c, ℓ_{d1}, and ℓ_{d2}.

1.4 Labels to Noninterference

We now have a more general version of noninterference specified by a lattice $(\Lambda, \sqsubseteq)$ of security levels. Some examples include



We now generalize $\sigma_1 =_{Low} \sigma_2$ to $\sigma_1 =_\ell \sigma_2$ for an arbitrary label ℓ.

$$\forall x.\Gamma(x) \sqsubseteq \ell \Leftrightarrow \sigma_1(x) =_\ell \sigma_2(x).$$

A program c is **noninterfering** if

$$\forall \sigma_1, \sigma_2, \sigma_1', \sigma_2', \ell \in \Lambda. \sigma_1 =_\ell \sigma_2 \wedge \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \Rightarrow \sigma_1' =_\ell \sigma_2'.$$

This definition is extensional, which means a temporary violation of noninterfering in intermediate steps does not imply the entire program is not noninterfering.

2 Threat Model

The threat model defines how an adversary interacts with the model. A stronger threat model allows adversaries more interactions with the model. Information is conveyed through information channels as categorized by Lampson (1973):

- **Legitimate channels** - files, console, network messages...
- **Covert channels** - execution time, heat emission, energy consumption...
  - **Side channels** are covert channels that can be exploited by a passive adversary simply observing the channel.

2.1 Termination Sensitivity

Our earlier definition of NI is termination-insensitive, assuming the attacker ignores all execution that do not terminate. Therefore we can modify our definition to a program c is **termination-sensitive noninterfering** if for all $\sigma_1, \sigma_2, \ell \in \Lambda$, $\sigma_1 =_\ell \sigma_2$ implies either

$$\exists \sigma_1', \sigma_2'.\ \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \wedge \sigma_1' =_\ell \sigma_2'$$

or both executions diverge.

2.2 Timing Sensitivity

The definition can be further expanded to take into account execution time in general with **time-sensitivity**. There are several ways to think about time:

- Number of computational steps - far removed from reality.
- External time - the "wall clock" time. This is hard to capture due to memory hierarchy and microarchitectural details.
- Internal time - such as information shared by threads on the same machine through ordering.

Computational steps and external time can be modeled by adding a counter variable to the state T that is observable by entities with low labels. Internal timing requires reasoning about concurrency.

2.3 Interaction

The "batched" model of computation does not capture an adversary providing input and observing output during the execution. The solution is to add input and output operations to **IMP**,

$$c ::= \ldots \mid x := \text{input from } \ell \mid \text{output } x \text{ to } \ell$$

The new semantics is

$$\langle c, \sigma \rangle \longrightarrow^{\tau} \langle c, \sigma' \rangle$$

where the trace τ is a sequence of events. We can expand the definition of noninterfering to take traces into account:

$$\begin{aligned} \forall \sigma_1, \sigma_2, \sigma_1', \sigma_2', \ell \in \Lambda.\ \sigma_1 =_\ell \sigma_2 \\ \wedge \langle c, \sigma_1 \rangle \longrightarrow^{\tau_1} \langle \text{skip}, \sigma_1' \rangle \\ \wedge \langle c, \sigma_2 \rangle \longrightarrow^{\tau_2} \langle \text{skip}, \sigma_2' \rangle \\ \wedge \mathit{inputs}(\tau_1) =_\ell \mathit{inputs}(\tau_2) \\ \Rightarrow \tau_1 =_\ell \tau_2. \end{aligned}$$

This states that if initial memories are ℓ-equivalent and low inputs are identical, then the traces are low-equivalent.

2.4 Progress Sensitivity

Traces allow us to define **progress sensitivity** (i.e. can the attacker observe whether the program is making progress?) as whether the attacker can observe the trace during execution. It is a generalization of termination sensitivity.

2.5 Program Code

Noninterference typically assumes the attacker knows the program code. Some other models also assume the attacker can provide code to the program (which can be simulated from attacker-provided input).

2.6 Computational Ability

We require a bound on the attacker's computational ability as cryptographic algorithms are only secure against an attacker with bound computational ability.

2.7 Views of a System

We can define the attacker's view on the system as a function from the system state (or history) to the attacker's observations. This allows us to model cases where the attacker can only observe a part of the system, such as in distributed systems. With views, a program c is **noninterfering** if

$$\forall \sigma_1, \sigma_2, \sigma_1', \sigma_2', \ell \in \Lambda.\ \sigma_1 =_\ell \sigma_2 \wedge \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \Rightarrow \mathit{view}(\sigma_1') =_\ell \mathit{view}(\sigma_2')$$

Kozyri et. al. summarizes the threat models as follows:

The adversary can:	**Example security conditions**
Observe termination	Termination-sensitive noninterference
Observe time	Time-sensitive noninterference
Observe output stream	Progress-sensitive noninterference
and provide input stream	Reactive noninterference, GMNI, noninference, generalized noninference
and use input strategies	Nondeducibility on strategies
and be a concurrently executed program	P_BNDC
Write program code	Noninterference against active adversary
Observe views of system behavior	Nondeducibility, Opaqueness

3 Computational Model

3.1 Nondeterminism

Noninterference doesn't hold for a nondeterministic system. For example,

$$\mathit{low} := 42 \,[]\, 7$$

does not satisfy noninterference because the way nondeterminism is resolved may depend on secret information, which is called a **refinement attack**. Thus we need the following definition: a program c is **generalized-noninterfering** if

$$\begin{aligned}&\forall \sigma_1, \sigma_2, \sigma_1', \sigma_2', \ell \in \Lambda, \sigma_1 =_\ell \sigma_2 \wedge \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \\ &\Rightarrow \exists \sigma_3, \sigma_3'\ s.t.\ \sigma_3 =_{\mathit{Low}} \sigma_1 \wedge \sigma_3 =_{\mathit{High}} \sigma_2 \wedge \langle c, \sigma_3 \rangle \Downarrow \sigma_3' \wedge \sigma_3' =_{\mathit{Low}} \sigma_1'.\end{aligned}$$

This represents the notion that secret inputs do not constrain the range of possible public outputs.

3.2 Observational Determinism

Alternatively, we use the notion of **observational determinism**. This requires the resolution of *Low* nondeterminism (e.g. processes picked by the scheduler) to not depend on secret information. The definition is the same as the deterministic one - a program c is noninterfering if

$$\forall \sigma_1, \sigma_2, \sigma_1', \sigma_2', \ell \in \Lambda.\ \sigma_1 =_\ell \sigma_2 \wedge \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \Rightarrow \sigma_1' =_\ell \sigma_2'.$$

While this eliminates refinement attacks, it also eliminates all public nondeterminism, which we may want in our program.

3.3 Probability

Possibilistic nondeterminism may not be sufficient if a program will most likely leak secret information. Therefore we define **Probabilistic** noninterference to be when the ***distribution*** of low outputs is independent of high inputs, using a probabilistic semantics. It assumes a probabilistic semantics $\langle c, \sigma_1 \rangle \Downarrow \mathfrak{D}$ where $\mathfrak{D}$ is a (sub-)distribution over stores:

$$\forall \sigma_1, \sigma_2, \mathfrak{D}_1, \mathfrak{D}_2, \ell \in \Lambda.\ \sigma_1 =_\ell \sigma_2 \wedge \langle c, \sigma_1 \rangle \Downarrow \mathfrak{D}_1 \wedge \langle c, \sigma_2 \rangle \Downarrow \mathfrak{D}_2 \Rightarrow \mathfrak{D}_1|_\ell = \mathfrak{D}_2|_\ell.$$

Here $\mathfrak{D}|_\ell$ means projecting to a distribution over the low-observable part of the store.

3.4 Concurrency

In a concurrent setting, information might flow by

- Interaction between threads - e.g. race conditions
- Scheduling
- Memory model
- Speculative execution - e.g. Spectre and Meltdown

However, no new definition of noninterference is necessary. We just need to extend the existing language and semantics to support concurrency.

3.5 Reclassification

Noninterference is too restrictive in practice. Sometimes we may need to declassify information, i.e. weakening confidentiality requirements, in cases like displaying the last 4 digits credit card number, and making a patient's record available to the assigned doctor. We may also need to erase information, in cases like deleting credit card information after a transaction or securing sensitive information when a submarine surfaces. Declassification has to be controlled as we do not we want to release confidential information at once.

There are "dimensions" of declassification such as what is declassified, to whom, and where in the system. An example mechanism include **delimited release** that allows declassification given a set of **escape hatch** expressions.

A program c and a set of escape hatches $\{a_1, \ldots, a_n\}$ satisfies delimited release if:

$$\begin{aligned}
&\forall \sigma_1, \sigma_2, \sigma_1', \sigma_2'. \\
&\sigma_1 =_{Low} \sigma_2 \wedge \langle c, \sigma_1 \rangle \Downarrow \sigma_1' \wedge \langle c, \sigma_2 \rangle \Downarrow \sigma_2' \\
&\wedge \forall i \in \{1, \ldots, n\}, \sigma_1(a_i) = \sigma_2(a_i) \\
&\Rightarrow \sigma_1' =_{Low} \sigma_2'
\end{aligned}$$

allowing the low-outputs to also depend on the escape hatch expressions. Note that the escape hatch expressions a_i may depend on $High$ variables, such as an average salary of individual secret salaries.

3.6 Quantitative Information Flow

Sometimes information leaks are unavoidable (i.e. side channels). **Quantitative information flow** can measure the magnitude of the leakage using information theory, by comparing the adversary's degree of uncertainty before and after the execution:

$$\text{info leakage} = \text{initial uncertainty - remaining uncertainty.}$$

There are different ways to measure leakage, such as Shannon entropy, Bayes vulnerability, Renyi's min entropy, gain functions, etc. Moreover, not all bits are equal - bits of the encryption key may be more important than bits of one's SSN.

As an example, using Shannon Entropy $H(X)$ and conditional entropy $H(X|Y)$, leakage is measured as

$$\text{leakage} = H(In_{secret}) - H(In_{secret} \mid In_{public}, Out_{public}).$$