Language-Based Security

Lecture 2: Information Flow Semantics

Stephen Chong, Harvard University
Road Map

• Intro
  • Formal Methods for Security
  • Language-Based Security
  • Case Study: Noninterference

• Primer on Computer Security

• Information Flow
  • Semantics
    • Enforcement
    • Beyond confidentiality

• Enforcing Language Abstractions
Sensitive Information

• Many systems handle a variety of sensitive information
• How do we ensure that the system is handling the information securely?
Access Control Isn’t Enough

- Access control can restrict who can access information.
- But it (typically) doesn’t restrict what happens to the information after access.
- If “handling information securely” means, e.g., only certain entities should learn about the information, then access control is close, but not exactly aligned.
Information Flow

• An **extensional** specification of information security
  • Define security in terms of the observable behavior of the system
  • Not in terms of the implementation details, such as code patterns, mechanisms, etc.
    • i.e., the “intension” of the system
• (Enforcement of an extensional security condition will, of course, depend on implementation details. We will examine enforcement of info flow later.)
Semantics of Information Flow
• Cohen (1976) introduced **strong dependency**
  • Essentially, the key definition of noninterference used today
• Intuition: information **flows** from one entity A to another entity B when B depends on or is influenced by A
• Definition: 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.
• Security is the absence of certain strong dependencies
Strong Dependency Example

• In the setting of IMP, with 2 security levels
• Context $\Gamma$ maps variables to \{Low, High\}
• Write $\sigma_1 =_{\text{Low}} \sigma_2$ if states $\sigma_1$ and $\sigma_2$ are equal on all low variables
  • For all $x$, if $\Gamma(x) = \text{Low}$ then $\sigma_1(x) = \sigma_2(x)$
• Definition: Program $c$ is **noninterfering** if:
  For all $\sigma_1$, $\sigma_2$, $\sigma'_1$, $\sigma'_2$,
  if $\sigma_1 =_{\text{Low}} \sigma_2$ and $\langle c, \sigma_1 \rangle \downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \downarrow \sigma'_2$
  then $\sigma'_1 =_{\text{Low}} \sigma'_2$
• i.e., no strong dependencies from high inputs to low outputs
Model of computer system

Reason about model

Real computer system
Beyond NonInterference for IMP

• In general, formulating an info flow property for a system involves choosing
  • The entities under consideration
    • e.g., who is involved, what’s an input, what’s an output, ...
  • The conditions under which flows between these entities are allowed or forbidden.

• Much research on info flow over the last 5 decades has considered focused on different threat models, computational models, and conditions that determine whether flows are allowed...
Beyond NonInterference for IMP

- **Computational model** indicates the entities that are manipulated during system executions.
- **Threat model** indicates the entities with which the adversaries interact.
- Specifying **allowed or forbidden flows** between entities amounts to stipulating allowed or forbidden flows between the system and the adversaries.
- We will explore the space of information flow properties by varying the computational model, the threat model, and the expressiveness of the conditions to specify restrictions on info flows.
Lecture Roadmap

- Labels and Flow Relations
  - Threat model
    - Termination, Timing, and Interaction
    - Computational ability
  - Computational model
    - Nondeterminism
    - Probability
    - Concurrency
  - Reclassification
    - Quantitative info flow
Labels

• Syntactic objects associated with entities of a system
  • E.g., Secret, Public
  • E.g., Trusted, Untrusted
  • E.g., Alice, Bob, Charlie, ...
  • E.g., (Level, Compartment) where
    Level ∈ { Public, Confidential, Secret, TopSecret} and
    Compartment ∈ { Nuclear, Cryptography, Biological, ...}

• Info-flow policy might described allowed (or forbidden) flows between entities based on labels

• Labels might have rich structure but don’t themselves describe policies
  • Labels represent restrictions on how associated entities can be used
Flow Relations

• Info flow policy often represented as flow relation $\sqsubseteq$ on a set $\Lambda$ of labels
  • If $\ell_1 \sqsubseteq \ell_2$ then info is allowed to flow from $\ell_1$ to $\ell_2$
• What structure should flow relation $\sqsubseteq$ have?
  • Reflexive, i.e., for all $\ell \in \Lambda$ we have $\ell \sqsubseteq \ell$
  • Transitive?
    • i.e., for all $\ell_1, \ell_2, \ell_3 \in \Lambda$, if $\ell_1 \sqsubseteq \ell_2$ and $\ell_2 \sqsubseteq \ell_3$ then $\ell_1 \sqsubseteq \ell_3$
  • Reflexive and transitive is a pre-order
• If we add antisymmetry, it is a partial order
• Denning (1978) argues for a join-semi-lattice relation
  • i.e., a **least-upper bound** operation $\sqcup$
    • Upper bound: $\forall \ell_1, \ell_2 \in \Lambda$, $\ell_1 \sqsubseteq \ell_1 \sqcup \ell_2$ and $\ell_2 \sqsubseteq \ell_1 \sqcup \ell_2$
    • Least upper bound: $\forall \ell_1, \ell_2, \ell_3 \in \Lambda$, if $\ell_1 \sqsubseteq \ell_3$ and $\ell_2 \sqsubseteq \ell_3$ then $\ell_1 \sqcup \ell_2 \sqsubseteq \ell_3$

• Why?

• Given data $a$ and $b$, labeled respectively $\ell_a$ and $\ell_b$

• What should be label of operation $a \oplus b$?
  • Should be upper bound
  • Should be least upper bound, otherwise the following may not work (where $\ell_{d1}$ and $\ell_{d2}$ are both upper bounds of $\ell_a$ and $\ell_b$)
    • $c = a \oplus b$; $d1 = c$; $d2 = c$
From Labels to NI

- Here is a more general version of noninterference:
  - Lattice $(\Lambda, \sqsubseteq)$ of security levels

```
\begin{align*}
\emptyset & \sqsubseteq \{Alice\} \sqsubseteq \{Bob\} \\
\{Alice\} & \sqsubseteq \{Alice,Bob\} \\
\{Bob\} & \sqsubseteq \{Bob,Chuck\} \\
\{Chuck\} & \sqsubseteq \{Alice,Chuck\} \\
\{Alice,Bob\} & \sqsubseteq \{Alice,Bob,Chuck\} \\
\end{align*}
```

- Confidential
- Public
• Here is a more general version of noninterference:
  • Lattice ($\Lambda$, $\sqsubseteq$) of security levels
  • Context $\Gamma$ is function from variables to $\Lambda$
  • Write $\sigma_1 =_{\ell} \sigma_2$ if states $\sigma_1$ and $\sigma_2$ are equal on all low variables: For all $x$, if $\Gamma(x) \sqsubseteq \ell$ then $\sigma_1(x) = \sigma_2(x)$
  • Definition: Program $c$ is **noninterfering** if:
    For all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2, \ell \in \Lambda$
    if $\sigma_1 =_{\ell} \sigma_2$ and $\langle c, \sigma_1 \rangle \downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \downarrow \sigma'_2$
    then $\sigma'_1 =_{\ell} \sigma'_2$
Lecture Roadmap

- Labels and Flow Relations
- Threat model
  - Termination, Timing, and Interaction
  - Computational ability
- Computational model
  - Nondeterminism
  - Probability
  - Concurrency
- Reclassification
  - Quantitative info flow
Threat Model

• How adversary interacts with system
• Stronger threat model $\rightarrow$ more interactions $\rightarrow$ more opportunities for information flow to/from adversary

• **Information channels** convey information
  • Lampson (1973) categorizes them as:
  • **legitimate channels** (e.g., files, console, network messages, ...) and
  • **covert channels** (e.g., execution time, heat emission, noise emission, resource exhaustion, power consumption, ...)
    • **side channels** are covert channels exploited by passive adversary who simply observes the channel
Termination

• Earlier definition of NI is termination-insensitive
  • Implicitly assumes that attacker ignores all executions that fail to terminate

• Definition: Program $c$ is noninterfering if:
  For all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2, \ell \in \Lambda$
  if $\sigma_1 =_\ell \sigma_2$ and $\langle c, \sigma_1 \rangle \Downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \Downarrow \sigma'_2$
  then $\sigma'_1 =_\ell \sigma'_2$

• So while $(\text{high} > 0)$ do skip satisfies noninterference
Termination-Sensitivity

- Can modify security condition to account for termination channel
  - Key idea: termination behavior is determined by low inputs
    - Either both executions terminate or both executions diverge
- Definition: Program $c$ is **termination-sensitive noninterfering** if:
  
  For all $\sigma_1, \sigma_2, \ell$, if $\sigma_1 =_{\ell} \sigma_2$ then either
  - exists $\sigma'_1, \sigma'_2$. $\langle c, \sigma_1 \rangle \downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \downarrow \sigma'_2$ and
    $\sigma'_1 =_{\ell} \sigma'_2$
  - or
  - both executions diverge
Can the adversary observe how long an execution takes?

- **Timing sensitivity**

Termination sensitivity is an extreme example of timing sensitivity

Several ways of thinking about timing

- Number of steps the computational model takes
  - But suffers from big gap between model and reality
- External timing (“Wall clock time”)
  - Actually very hard to capture accurately in a model, as it depends on many low-level system details
    - Memory hierarchy, microarchitecture details, ...
  - Conceptually, can add new variable to state, T, which increases during execution and is low-observable

- Internal timing
  - E.g., thread running in the same system that can detect which event happens first
  - Concurrency (see later)
Interaction

• So far we have used a “batch”-like model of computation
  • Systems gets input, does all computation and produces output on termination

• Most systems are interactive
  • Adversary may make observations during executions
  • Adversary (and others) may provide inputs during execution

• Requires different computational model to express
Interaction

• Assume IMP with \( x := \text{input from } \ell \) and output \( x \) to \( \ell \)
• Semantics \( \langle c, \sigma \rangle \xrightarrow{\tau} \langle c', \sigma' \rangle \) where trace \( \tau \) is a sequence of events
  • \( \tau ::= \varepsilon \mid \tau \cdot \text{in}(n, \ell) \mid \tau \cdot \text{out}(n, \ell) \)
  • Intuitively: \( \langle c, \sigma \rangle \) takes one or more steps to \( \langle c', \sigma' \rangle \) producing trace \( \tau \)
• Interactive noninterference: if initial memories are low equivalent and low inputs are identical, then the traces are low-equivalent (i.e., low inputs and outputs are the same)
• Definition: Program \( c \) is noninterfering if:
  For all \( \sigma_1, \sigma_2, \sigma'_1, \sigma'_2, \tau_1, \tau_2, \ell \in \Lambda \)
  if \( \langle c, \sigma_1 \rangle \xrightarrow{\tau_1} \langle \text{skip}, \sigma'_1 \rangle \) and \( \langle c, \sigma_2 \rangle \xrightarrow{\tau_2} \langle \text{skip}, \sigma'_2 \rangle \)
  and \( \sigma_1 =\ell \sigma_2 \) and inputs(\( \tau_1 \)) =\( \ell \) inputs(\( \tau_2 \))
  then \( \tau_1 =\ell \tau_2 \)
Progress Sensitivity

• Can the attacker observe whether program is making progress (i.e., will produce another event)?

• Analogous to termination sensitivity, but for non-batch programs
Program Code

• Does the attacker know the code? Can they modify/provide code?

• Noninterference typically (implicitly) assumes attacker knows code

• Definition: Program $c$ is **noninterfering** if:
  
  For all $\sigma_1$, $\sigma_2$, $\sigma'_1$, $\sigma'_2$, $\ell$
  
  if $\sigma_1 =_{\ell} \sigma_2$ and $\langle c, \sigma_1 \rangle \Downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \Downarrow \sigma'_2$
  
  then $\sigma'_1 =_{\ell} \sigma'_2$

• Some models allow attacker to provide code (but this can typically be simulated by any attacker-provided input)
Attacker’s Computational Ability

- What can the attacker compute?
- E.g., does the following satisfy noninterference?

\[
\begin{align*}
\Gamma(msg1) &= \text{Low} \\
\Gamma(msg2) &= \Gamma(key) = \text{High} \\
\text{output } \text{encrypt}(msg1, \text{key}) &\text{ to Low} \\
\text{output } msg1 &\text{ to Low} \\
\text{output } \text{encrypt}(msg2, \text{key}) &\text{ to Low}
\end{align*}
\]

- Some versions of noninterference assume computational limits on attacker
Views of a System

• More generally, may define what the attacker can observe as a view of the system, a function from the system state (or history) to the attacker’s observations

• E.g., attacker sees a subset of the state of the system
  • Appropriate for a distributed system where some machines are compromised

• E.g., attacker sees power consumption of system

• Definition: Program c is noninterfering if:
  For all $\sigma_1, \sigma_2, \tau_1, \tau_2, \ell \in \Lambda$
  
  if $\sigma_1 = \ell \sigma_2$ and $\langle c, \sigma_1 \rangle \Downarrow \tau_1$ and $\langle c, \sigma_2 \rangle \Downarrow \tau_2$
  
  then $\text{view}(\ell, \tau_1) = \text{view}(\ell, \tau_2)$

• (Haven’t defined relation $\langle c, \sigma \rangle \Downarrow \tau$. Think of $\tau$ of as being the history of the computation, includes events, states, ..., anything we want to model as observable)
Threat Model Summary

- Many different versions of non-interference handle different threat models
- From Kozyri et al.

<table>
<thead>
<tr>
<th>The adversary can:</th>
<th>Example security conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observe termination</td>
<td>Termination-sensitive noninterference</td>
</tr>
<tr>
<td>Observe time</td>
<td>Time-sensitive noninterference</td>
</tr>
<tr>
<td>Observe output stream</td>
<td>Progress-sensitive noninterference</td>
</tr>
<tr>
<td>and provide input stream</td>
<td>Reactive noninterference, GMNI, non-inference, generalized noninference</td>
</tr>
<tr>
<td>and use input strategies</td>
<td>Nondeducibility on strategies</td>
</tr>
<tr>
<td>and be a concurrently executed program</td>
<td>P_BNDC</td>
</tr>
<tr>
<td>Write program code</td>
<td>Noninterference against active adversary</td>
</tr>
<tr>
<td>Observe views of system behavior</td>
<td>Nondeducibility, Opaqueness</td>
</tr>
</tbody>
</table>
Lecture Roadmap

- Labels and Flow Relations
- Threat model
  - Termination, Timing, and Interaction
  - Computational ability
- Computational model
  - Nondeterminism
  - Probability
  - Concurrency
- Reclassification
  - Quantitative info flow
Computational Model

• Computational model abstracts system functionality
  • Tightly coupled with threat model

• Computational model captures implementation details of a system, at varying levels of faithfulness
Nondeterminism

• So far we have considered deterministic systems
• Noninterference doesn’t hold for nondeterministic system
• E.g., with nondeterministic choice operator $a_1 \triangleright a_2$, program
  \[
  \text{low} := 42 \triangleright 7
  \]
  may not satisfy NI
• Intuitively, we don’t know how nondeterminism is resolved; may depend on secret information
  • So-called refinement attack
Generalized Noninterference

- Intuition: secret inputs do not constrain public outputs
  - i.e., all possible Low behaviors are possible with any High inputs

- Definition: Program c satisfies \textbf{generalized noninterference} if:

  For all $\sigma_1, \sigma_2, \sigma'_1, \sigma'_2$,
  if $\sigma_1 =_{\text{Low}} \sigma_2$ and $\langle c, \sigma_1 \rangle \downarrow \sigma'_1$ and $\langle c, \sigma_2 \rangle \downarrow \sigma'_2$

  then there exists $\sigma_3, \sigma'_3$ such that
  $\sigma_3 =_{\text{Low}} \sigma_1$ and $\sigma_3 =_{\text{High}} \sigma_2$ and
  $\langle c, \sigma_3 \rangle \downarrow \sigma'_3$ and $\sigma'_3 =_{\text{Low}} \sigma'_1$
Observational Determinism

- But resolution of nondeterminism is useful!
- **Observational determinism** requires that resolution of Low nondeterminism does not depend on secret information
  - E.g., if nondeterminism is due to scheduler choices of threads/processes, the scheduler should not depend on high information
- Definition same as deterministic NI! i.e., low view is determined by low inputs
  - For all $\sigma_1, \sigma_2, \tau_1, \tau_2, \ell \in \Lambda$
    - if $\sigma_1 =_{\ell} \sigma_2$ and $\langle c, \sigma_1 \rangle \downarrow \tau_1$ and $\langle c, \sigma_2 \rangle \downarrow \tau_2$
      - then $\text{view}(\ell, \tau_1) = \text{view}(\ell, \tau_2)$
- Pro: not subject to refinement attack
- Con: allows no public nondeterminism
• **Possibilistic** nondeterminism may not sufficiently model information flows if some choices are unlikely
  
  • E.g., \( a := 0 \uparrow 1 \uparrow \ldots \uparrow 999; \)
    
    \[
    \begin{align*}
    \text{if } (a = 0) \text{ then } low &:= 0 \uparrow 1 \\
    \text{else } low &:= \text{high mod } 2
    \end{align*}
    \]

• **Probabilistic** noninterference requires that the *distribution* of low outputs is independent of high inputs

• Assume probabilistic semantics \( \langle c, \sigma \rangle \downarrow \mathcal{D} \) where \( \mathcal{D} \) is a (sub-)distribution over stores

  • Add your favorite probabilistic operators to the language

• **Probabilistic Noninterference:** For all \( \sigma_1, \sigma_2, \mathcal{D}_1, \mathcal{D}_2, \ell \in \Lambda \)

  \[
  \begin{align*}
  \text{if } \sigma_1 =_{\ell} \sigma_2 \text{ and } \langle c, \sigma_1 \rangle \downarrow \mathcal{D}_1 \text{ and } \langle c, \sigma_2 \rangle \downarrow \mathcal{D}_2 \\
  \text{then } \mathcal{D}_1|_{\ell} = \mathcal{D}_2|_{\ell}
  \end{align*}
  \]

  • (Where \( \mathcal{D}|_{\ell} \) projects the distribution over stores to a distribution over the low-observable part of the store)
Concurrency

• When modeling concurrency, information might flow by
  • Interaction between threads
    • E.g., race conditions are a source of nondeterminism
  • Scheduling choices
  • Memory model
    • Sequential consistency, Total Store Order, Partial Store Order, ...

• Relatedly, speculative execution is source of real information leaks
  • E.g., Spectre and Meltdown attacks

• Don’t really need a new definition of noninterference
  • Other than extending our language and semantics to support concurrency
Lecture Roadmap

- Labels and Flow Relations
- Threat model
  - Termination, Timing, and Interaction
  - Computational ability
- Computational model
  - Nondeterminism
  - Probability
  - Concurrency
- Reclassification
  - Quantitative info flow
Reclassification

• In practice, noninterference is too restrictive
  • Information does not keep the same label throughout execution

• May need to declassify information
  • i.e., weaken confidentiality requirements
  • e.g., credit card number is confidential, but last 4 digits can be printed on receipt
  • e.g., when a physician is assigned to a patient, they can see the patient’s records
  • e.g., after a sealed-bid auction is concluded, the confidential bids may be made publics

• May need to erase information
  • i.e., strengthen confidentiality requirements
  • e.g., after transaction, merchant should no longer hold credit card information
  • e.g., when submarine surfaces, sensitive information should be encrypted
• How to declassify in a controlled way?
  • Don’t want to allow all confidential information to be released!
• Sabelfeld and Sands (2009) describe “dimensions” of declassification:
  • *What* info is declassified
  • *Who* declassifies the info
  • *Where* in the system (i.e., component) or label relation does the declassification
  • *When* (under what conditions) does declassification happen?
Example: Escape Hatches

- Delimited Release (Sabelfeld and Myers, 2003)
- Intuition: specifies what information may be declassified by a set of escape hatch expressions
- Definition: Program c and set of escape hatches \{a_1, ..., a_n\} satisfies delimited release if:
  
  For all \(\sigma_1, \sigma_2, \sigma'_1, \sigma'_2\),
  
  if \(\sigma_1 =_{\text{Low}} \sigma_2\) and \(\langle c, \sigma_1 \rangle \downarrow \sigma'_1\) and \(\langle c, \sigma_2 \rangle \downarrow \sigma'_2\)
  
  and for all \(i \in 1..n\), \(\sigma_1(a_i) = \sigma_2(a_i)\)

  then \(\sigma'_1 =_{\text{Low}} \sigma'_2\)
Example: Intransitive NI

• Intuition: remove transitivity as a requirement for the flow relation

• E.g.,

Typically a **trusted component** is the only component that is permitted to use intransitive flow relations (a form of *where* declassification)

• Security conditions might need to consider some of the implementation details to express this...
There are info leaks that are undesirable but unavoidable (e.g., via side channels)

How to understand the magnitude of these leaks?

Quantitative information flow uses information theory to measure leakage

Basic idea: info leakage = initial uncertainty – remaining uncertainty
Quantitative Info Flow

• Different ways of measuring leakage, e.g.,
  • Shannon entropy
  • Bayes vulnerability
  • Renyi’s min-entropy
  • Not all bits are equal: gain functions can capture value of bits

• E.g., Shannon entropy
  • For random variable $X$, $H(X)$ is the Shannon entropy of $X$
    • Expected number of bits to optimally encode value of $X$
    • Uncertainty or surprise in $X$

$$H(X) = - \sum_{x \in \mathcal{X}} \Pr(x) \log_2 \left( \Pr(x) \right)$$

• Conditional entropy $H(X|Y)$ information in $X$ given knowledge of $Y$

$$H(X|Y) = \sum_{y \in \mathcal{Y}} \Pr(Y = y) H(X|Y = y)$$

• Leakage = $H(\text{In}_{\text{Secret}}) - H(\text{In}_{\text{Secret}} \mid \text{In}_{\text{Public}}, \text{Out}_{\text{Public}})$
Selected References


