Introduction to Information Flow

CS 711
17 Sep 03
Andrew Myers

Lampson, 1973
Identifies difficulty of confining information to a process [actually a reprint of an earlier note]
- Problem later called information flow control
- Confinement is easy if you are draconian, but...
- Storage channels: explicit information transmission (writes to sockets, files, assignments)
- Covert channels: transmit by mechanisms not intended for signaling information (system load, run time, locks)
- Too optimistic about masking covert channels

Bell and LaPadula, 1973
- An abstract model intended to control information flow
  - Objects have a security level (e.g., unclassified, classified, secret, top secret)
  - Subjects (think: principals, processes) have a level
  - Subjects cannot read objects at a higher level (simple security property)
  - Subjects cannot write objects at a lower level (*-property, confinement property)
- Coarse-grained
- Multics/AIM ring model
  - Doesn’t help users...

Generalizing levels to lattices
[Denning, 1976]
- Security levels may in general form a lattice (or just a partial order)
- $L_1 \leq L_2$ means information can flow from level $L_1$ to level $L_2$
  - $L_2$ describes greater confidentiality requirements
- Lattice supports reasoning about information channels that merge and split
  ($\cup = \text{LUB}, \cap = \text{GLB}$)
  \[
  c := a + b \\
  a,b := c \cup \text{a} \cap \text{b}
  \]

Multilevel security policies
[Feiertag et al., 1977]
- Security level is a pair $(A, C)$ where $A$ is from a totally ordered set (unclassified, ...) and $C$ is a set of categories
- Example: $(\text{secret}, \{\text{nuclear}\}) \subseteq (\text{top secret}, \{\text{nuclear, iraq}\})$ but $\notin (\text{secret}, \{\text{iraq}\})$
  \[
  (A_1, C_1) \subseteq (A_2, C_2) \iff A_1 \leq A_2 \land C_1 \subseteq C_2
  \]

Integrity
[Neumann et al., 1976; Biba, 1977]
- Integrity can also be described as a label
  - Prevent: bad data from affecting good data
- $L_1 \leq L_2$ means information can flow from level $L_1$ to level $L_2$
  - $L_2$ describes lower integrity requirements
- Integrity is dual to confidentiality

Increasing confidentiality
Decreasng integrity

Increasing security
Less readable
Less available
Less safe
Less secure
More readable
More available
More safe
More secure
Mandatory access control

- Controlling information flow with dynamic mechanisms ala Bell-LaPadula
- Processes that read higher level information may have their level increased to prevent them from leaking it
  - Label creep
- Single-level channels vs. multilevel channels
  - Single-level channels check
  - Multilevel channels explicitly label outgoing data

Implicit flows

- Covert storage channels arising from control flow. Example:
  ```java
  boolean b := <some secret>
  if (b) {
    x = true; f();
  }
  ```
- Creates information flow from b to x, need to enforce $L_b \subseteq L_x$
- Run-time check requires whole process labeled secret after branch

Static analysis of information flow

[Denning & Denning, 1977]

- Inference algorithm for determining whether variables are high or low
- Program-counter label tracks implicit flows
  - Computed by dataflow analysis
    
    ```plaintext
    pc = ⊥
    boolean b := <some secret>
    if (b) {
      x = true; f();
    }
    pc = L_b
    pc = L_x
    ```

Noninterference

[Cohen, 1977][Goguen & Meseguer, 1982]

- Inputs only affect outputs higher in the lattice
- An end-to-end, semantic definition of security

A formalization

- Key idea: behaviors of the system $C$ don’t reveal more information than the low inputs
- Consider applying $C$ to inputs $x$. Define:
  - $[C]$, the result of $C$ applied to input $x$
  - $x_1 \rightarrow_{s_1} s_2$ means inputs $s_1$ and $s_2$ are indistinguishable to the low user at level $L$. E.g., $(H,L) \approx (H',L)$
  - $[C] x_1 \approx_{s_1} [C] x_2$, means results are indistinguishable: low view relation captures observational power

Noninterference for $C$: $x_1 \rightarrow_{s_1} s_2 \Rightarrow [C] x_1 \approx_{s_1} [C] x_2$

“Low observer doesn’t learn anything new”

Unwinding condition

- Induction hypothesis for proving noninterference
- Assume $[C]$ defined by a transition relation $s \rightarrow s'$
  ```plaintext
  s_1 \rightarrow \frac{h}{s_1'} \quad s_2 \rightarrow \frac{h}{s_2'}
  ```
- Each step of execution preserves equivalence
- By induction: whole trace preserves equivalence, equivalence inputs produce equivalent results
- $\approx$ must be an equivalence—need transitivity
Example
• “System” is a program with a memory
  \[
  \text{if } h_1 \text{ then } h_2 := 0 \quad \text{else } h_2 := 1; \quad l := 1
  \]
• Define: \[ s = \langle c, m \rangle \]
• Define: \[ \langle c_1, m_1 \rangle \approx \langle c_2, m_2 \rangle \text{ if identical after:} \]
  - erasing high pc terms from \( c \)
  - erasing high memory locations from \( m \)
• Choice of \( \approx \) controls what low observer can see at a moment in time
• Current command \( c \) included in state to allow proof by induction

Termination sensitivity
Is this program secure?
\[
\text{while } h > 0 \text{ do } h := h+1; \quad l := 1
\]
\[
\begin{align*}
  \{ h \rightarrow 0, l \rightarrow 0 \} & \rightarrow^* \{ h \rightarrow 0, l \rightarrow 1 \} \\
  \{ h \rightarrow 1, l \rightarrow 0 \} & \rightarrow^* \{ h \rightarrow 1, l \rightarrow 0 \} \quad (\forall i > 0)
\end{align*}
\]
• Low observer learns value of \( h \) by observing nontermination, change to \( l \)
• But… might want to ignore this channel to make analysis feasible

Low views
• Low view relation \( \approx_L \) on traces modulo \( \approx \) determines ability of attacker to observe system execution
• Termination-sensitive but no ability to see intermediate states:
  \[
  \langle s_1, s_2, \ldots, s_m \rangle \approx \langle s_{\ell_1}, s_{\ell_2}, \ldots, s_{\ell_n} \rangle \text{ if } s_m \approx_L s_{\ell_n} \]
  \& all infinite traces are related by \( \approx \)
• Termination-insensitive:
  \[
  \langle s_1, s_2, \ldots, s_m \rangle \approx \langle s_{\ell_1}, s_{\ell_2}, \ldots, s_{\ell_n} \rangle \text{ if } s_m \approx_L s_{\ell_n} \]
  \& infinite traces are related by \( \approx \) to all traces
• Timing-sensitive:
  \[
  \langle s_1, s_2, \ldots, s_m \rangle \approx \langle s_{\ell_1}, s_{\ell_2}, \ldots, s_{\ell_n} \rangle \text{ if } s_m \approx_L s_{\ell_n} \]
  \& all infinite traces are related by \( \approx \)
• Not always an equivalence relation!

Security specifications
• Is security proving that a program is correct?
• Ordinary correctness specifications:
  \[
  \{ P \} S \{ Q \}
  \]
  precondition \( P \) \( \Rightarrow \) postcondition \( Q \)
• How do we know the specification satisfies security requirements?
• Example:
  - Precondition: all salaries in the database are nonnegative
  - Postcondition: \( x \) contains the average salary
• Partial correctness assertions describe properties satisfies by every execution individually; information flow assertions compare every pair of executions