Introduction to Information Flow

CS 711
17 Sep 03
Andrew Myers

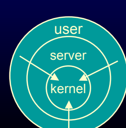
Lampson, 1973

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)
- $L_1 \sqsubseteq L_2$ 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| ($\sqcup$=LUB, $\sqcap$=GLB)

c := a + b $\quad L_a \sqcup L_b \sqsubseteq L_c$

a,b := c $\quad L_c \sqsubseteq L_a \sqcap L_b$

Multilevel security policies

[Feiertag et al., 1977]

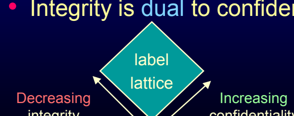
- Security level is a pair (A,C) where A is from a totally ordered set (unclassified, …) and C is a set of *categories*
- Example: (secret, {nuclear}) $\sqsubseteq$ (top secret, {nuclear, iraq}) but $\not\sqsubseteq$ (secret, {iraq})

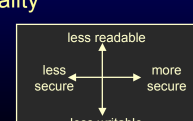
$$(A_1,C_1)\sqsubseteq(A_2,C_2) \text{ iff } A_1 \le A_2 \;\&\; C_1 \subseteq C_2$$

Integrity

[Neumann et al., 1976; Biba, 1977]

- Integrity can also be described as a label
- Prevent: bad data from affecting good data
- $L_1 \sqsubseteq L_2$ means information can flow from level L_1 to level L_2
  - L_2 describes lower integrity requirements
- Integrity is dual to confidentiality





Mandatory access control

- Department of Defense "Orange Book" (a.k.a. *DoD Trusted Computer System Evaluation Criteria*, 1985)
- 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:

```
boolean b := <some secret>
if (b) {
  x = true; f();
}
```

- Creates information flow from b to x, need to enforce $L_b \sqsubseteq L_x$
- Run-time check requires whole process labeled secret after branch

Static analysis of information flow

[Denning & Denning, 1977]

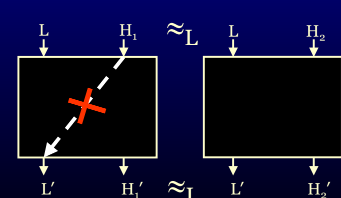
- Inference algorithm for determining whether variables are high or low
- Program-counter label tracks implicit flows
  - Computed by dataflow analysis

```
pc = ⊥  →
           boolean b := <some secret>
           if (b) {
pc = Lb →    x = true; f();
           }
pc = ⊥  →
```


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 s. Define:

  $[\![C]\!]\, s$ is the result of C applied to input s

  $s_1 =_L s_2$ means inputs s_1 and s_2 are indistinguishable to the low user at level L. E.g., (H,L) $\approx_L$ (H′,L)

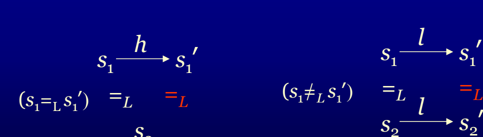
  $[\![C]\!]s_1 \approx_L [\![C]\!]s_2$ means results are indistinguishable : low view relation captures observational power

Noninterference for C: $s_1 =_L s_2 \;\Rightarrow\; [\![C]\!]s_1 \approx_L [\![C]\!]s_2$

"Low observer doesn't learn anything new"

Unwinding condition

- Induction hypothesis for proving noninterference
- Assume $[\![C]\!]$ defined by a transition relation $s \rightarrow s'$



- Each step of execution preserves equivalence
- By induction: whole trace preserves equivalence, equivalence inputs produce equivalent results
- $=_L$ must be an equivalence—need transitivity

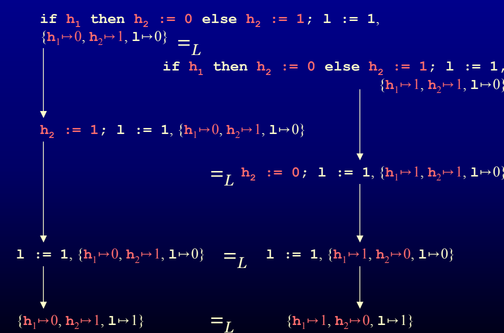
Example

- “System” is a program with a memory

```
if h1 then h2:= 0
        else h2:= 1;
l := 1
```

- Define: $s = \langle c, m\rangle$
- Define: $\langle c_1, m_1\rangle =_L \langle c_2, m_2\rangle$ if identical after:
  - erasing high pc terms from c_i
  - erasing high memory locations from m_i
- Choice of $=_L$ controls what low observer can see at a moment in time
- Current command *c* included in state to allow proof by induction

Example



Termination sensitivity

Is this program secure?

```
while h > 0 do h := h+1;
l := 1
```

$$\{h \mapsto 0, l \mapsto 0\} \longrightarrow^* \{h \mapsto 0, l \mapsto 1\}$$

$$\{h \mapsto 1, l \mapsto 0\} \longrightarrow^* \{h \mapsto i, l \mapsto 0\} \quad (\forall i > 0)$$

- 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 $=_L$ determines ability of attacker to observe system execution
- Termination-sensitive but no ability to see intermediate states:
  $(s_1, s_2, \ldots, s_m) \approx_L (s'_1, s'_2, \ldots s'_n)$ if $s_m =_L s_n'$
  & all infinite traces are related by $\approx_L$
- Termination-insensitive:
  $(s_1, s_2, \ldots, s_m) \approx_L (s'_1, s'_2, \ldots s'_n)$ if $s_m =_L s_n'$
  & infinite traces are related by $\approx_L$ to all traces
- Timing-sensitive:
  $(s_1, s_2, \ldots, s_n) \approx_L (s'_1, s'_2, \ldots s'_n)$ if $s_n =_L s_n'$
  & all infinite traces are related by $\approx_L$
- Not always an equivalence relation!

Security specifications

- Is security proving that a program is correct?
- Ordinary correctness specifications:

  {P} S {Q}

  precondition P ➔ 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