CS590U
Access Control: Theory and Practice

Lecture 3 (Jan 18)

State Transition Systems & The Graham-Denning Schemes

Announcements

- Mailing list
  - CS590U_Spring2005@cs.purdue.edu
  - To join: send email to mailer@cs.purdue.edu
    - with the following in the email body
      add your_email to CS590U_Spring2005
  - You should have received a note from the mailing list
- HW1 due today
- Project pre-proposal due on Thursday

The Need For A Formal Model of The System

- Need to describe the things we want to study and analyze the security properties of them
  - analyzing security properties
  - comparing expressive powers
- What systems to model?
  - computer systems
  - protection systems
- How to model a system?

Example

- A coffee vending machine that accepts nickle, dime, quarter and gives out one coffer (cost 10 cents) and changes
- Goal: show that a design (or an implementation) satisfies various properties, e.g.,
  - never gives a coffee for less than 10 cents
  - never takes more money from a user
  - never frustrates a user (whatever that means)

Kripke Structures

- Let AP be a set of atomic propositions. A Kripke structure M over AP is a four-tuple
  - S is finite set of states
  - $S_0 \subseteq S$ is the set of initial states
  - $R \subseteq S \times S$ is a transition relation
  - $L: S \to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state
- Often times, R is required to be total
    - $\forall s\ \exists s'\ (s,s') \in R$

Usage of Kripke Structures

- Given a Kripke structure $\langle S, S_0, R, L \rangle$, a path is an infinite sequence $s_0, s_1, \ldots$ of states such that $s_0 \in S_0$ and $(s_i, s_{i+1}) \in R$
- Verifying properties
  - A property may be specified in a temporal logical formula on paths and propositional variables on each state
- Showing that two Kripke structure are equivalent under some definition of "equivalence"

Questions to Think?

- How to use Kripke structure to model the coffee vending machine?
- Is the Kripke structure sufficient (or convenient) for modelling the coffee vending machine?

Coffee Machine:

- Let AP={coffee, change}
  - S: {0, 5, 10, 15, 25, 30}
  - S_0: {0}
  - R: (0,0), (0,5), (0,10), (0,25), (5,10), (5,15), (10,0), (15,0), (25,0), (30,0)
  - L:
    - 0: coffee is false, change is 0
    - 5: coffee is false, change is 0
    - 10: coffee is true, change is 0
    - 15: coffee is true, change is 5
    - 20: coffee is true, change is 10 ...

Issues in Modelling

- Granularity of state transitions
  - too coarse (may miss problems)
  - too fine-grained (may find false problems)

Modeling Reactive Systems

- A system changes states as a result of external actions
- These results may cause certain outputs
  - e.g., “yes, access is allowed”, “no, access is denied”, etc.
- Need to model external actions & outputs

Labelled State Transition Systems

- Each state-transition is labeled with a label
  - intuition: an action
- Not entirely clear about how to model an output.
  - one possibility: as another action
- Security properties will need to be specified using information on labels and outputs
- May need a new theory (or at least) substantial extensions to existing theory

The Access Matrix Model

History

- Lampson'1971
  - "Protection"
- Refined by Graham and Denning'1972
  - "Protection---Principles and Practice"
- Harrison, Ruzzo, and Ullman'1976
  - "Protection in Operating Systems"

Access Matrix

- A set of subjects S
- A set of objects O
- A set of rights R
- An access control matrix
  - one row for each subject
  - one column for each subject/object
  - elements are right of subject on another subject or object

The Graham-Denning Work

- Based on access matrices
- Focuses on access control within an operating system
- Explores various possibilities of discretionary access control

Seven Levels of Protection / Separation

1. No sharing at all
2. Sharing copies of programs or data files
3. Sharing originals of programs or data files
4. Sharing programming systems or subsystems
5. Permitting the cooperation of mutually suspicious subsystems, e.g., debugging or proprietary subsystems
6. Providing memory-less subsystems
7. Providing "certified" subsystems

Elements in Graham-Denning

- Objects: have unique identifier
- Subjects
  - a subject is a pair (process, domain)
  - forging a subject identifier is impossible (authentication)
- Protection state
  - modeled using an access matrix (can also be represented as a graph)
- No modeling of actual accesses (only access permissions)
  - whether this is sufficient depends on the properties to be studied

Special Rights in Graham-Denning Model

- Each subject/object has an owner
- Each subject has a controller (which may be itself)
- A right may be transferable or nontransferable

	Objects					
Subjects	S_1	S_2	S_3	O_1	O_2	O_3
S_1	control			owner	read write	
S_2		control	read*			execute
S_3			control		owner	

Eight Commands in Graham-Denning Model

1. subject x creates object o
   - no precondition
   - add column for o
   - place `owner’ in A[x,o]
2. subject x creates subject s
   - no precondition
   - add row and column for s
   - place control, `owner’ in A[x,s]

Eight Commands in Graham-Denning Model

3. subject x destroys object o
   - precondition: `owner’ in A[x,o]
   - delete column o

4. subject x destroys subject s
   - precondition: `owner’ in A[x,s]
   - delete row and column for s

Eight Commands in Graham-Denning Model

5. subject x grants a right r/r* on object o to subject s
   - precondition: `owner’ in A[x,o]
   - stores r/r* in A[s,o]
6. subject x transfers a right r/r* on object o to subject s
   - precondition: r* in A[x,o]
   - stores r/r* in A[s,o]

Eight Commands in Graham-Denning Model

7. subject x deletes right r/r* on object o from subject s
   - precondition: `control’ in A[x,s] or `owner’ in A[x,o]
   - delete r/r* from A[s,o]

Eight Commands in Graham-Denning Model

8. subject x checks what rights subject s has on object o [w := read s,o]
   - precondition: `control' in A[x,s] OR `owner' in A[x,o]
   - copy A[s,o] to w
- This does not affect the protection state.
   - policy review functions
   - useful when analyzing external behaviors of the protection system, not clear why needed in this paper

Messy Details

- Some requirements place additional constraints on state-transitions
  - Each subject is owner or controlled by at most one other subject
    - cannot transfer/grant owner right
  - It is undesirable for a subject to be `owner' of itself, for then it can delete other subjects' access to itself
  - [The relation "owner" defines naturally a tree hierarchy on subjects.]
    - What does it take to maintain the hierarchy?

Other possible extensions

- Transfer-only copy flags
- Limited-use access attributes
  - needs to model access
- Allow a subject to obtain a right that its subordinate has.
- The notion of "indirect" right
  - S_2 has indirect right over S means that S_2 can access anything that S is allowed to access, but S_2 cann't take right from S
  - differs from basic notion of an access matrix

How to Analyze Security Properties?

- "To prove that a protection model, or an implementation of it, is correct, one must show that a subject can never access an object except in an unauthorized manner"
  - any action by a subject cannot be an *authorized* access
  - any action that changes the protection state cannot lead to a new state in which some subject has *unauthorized* access

Issues of Trust

- Trusted vs. trustworthy
  - minimize trusted things
  - maximize trustworthy things
- A subject who has read* to an object can grant read to anyone
  - such a subject often needs to be trusted
  - similar issue: multiple owners of an object
- Someone having read access to an object can make copies of the object: read = read*

Approaches to the Trust Issue

- Trust human users, but not subjects
- Enable the analysis and understanding of trust
  - for a particular security property, who are trusted?
  - example: simple safety analysis [(o,r)-safety]
    - whether in a future state, a particular subject can get access to a particular object

Simple Safety Analysis in Graham-Denning

1 Subroutine isSafeGD ($\gamma, \psi, \omega, \mathcal{T}$)
2 /* inputs: γ, ψ, $\omega = \langle s, o, x \rangle$, $\mathcal{T} \subseteq \mathcal{S}$ */
3 /* output: true or false */
4 if $x \in \mathcal{R}_b^*$ then let $y \leftarrow x$
5 else if $x \neq \mathit{own} \wedge x \neq \mathit{control}$ then let $y \leftarrow x^*$
6 else let $y \leftarrow \mathit{invalid}$ /* No copy flags for *own* or *control* */
7 if $x \notin R_\psi$ then return true
8 if $x = \mathit{control} \wedge o \in \mathcal{O} - \mathcal{S}$ then return true
9 if $x \in M_\gamma[s, o]$ then return false
10 if $y \in M_\gamma[s, o]$ then return false
11 if $\mathcal{T} \supseteq S_\gamma$ then return true
12 if $o \notin O_\gamma$ then return false
13 if $\exists \widehat{s} \in S_\gamma - \mathcal{T}$ such that $y \in M_\gamma[\widehat{s}, o]$ then return false
14 for each sequence $\mathcal{U}, s_n, \ldots, s_2, s_1$ such that
15 $\mathit{own} \in M_\gamma[s_1, o] \wedge \cdots \wedge \mathit{own} \in M_\gamma[s_n, s_{n-1}] \wedge \mathit{own} \in M_\gamma[\mathcal{U}, s_n]$ do
16 if $\exists s_i \in \{s_1, \ldots, s_n\}$ such that $s_i \in S_\gamma - \mathcal{T}$ then return false
17 return true

Figure 2: The subroutine isSafeGD returns "true" if the system based on the Graham-Denning scheme, characterized by the start-state, γ, and state-change rule, ψ, satisfies the safety property with respect to ω and $\mathcal{T}$. Otherwise, it returns "false". In line 6, we assign some invalid value to y, as there is not corresponding right with the copy flag for the rights *own* and *control*. In this case, the algorithm will not return in line 10 or 13.

Implementation Issues

- Storing the access matrix
  - by rows: capability lists
  - by column: access control lists
  - through indirection:
    - e.g., key and lock list
    - e.g., groups, roles, multiple level of indirections, multiple locks
- How to do indirection correctly and conveniently is the key to management of access control.

An Open Problem

- There are many possibilities in the Graham-Denning approach to Discretionary Access Control
- How to abstract a scheme out of these possibilities so that
  - each possibility is an individual instance
  - properties of the scheme can be analyzed

The Bell-LaPadula Model of Computer Systems

- Basic elements:
  - subjects — S
  - objects — O
  - security labels — a partially-ordered set $\langle L, \leq \rangle$
  - access rights:

e	execute	(no read/no write)
r	read	(read only)
a	append	(write only)
w	write	(read/write)

The Bell-LaPadula Model of Computer Systems

- A system state is denoted by a triple
  - b: the current access set, a set of triples (subject, object, access-attribute)
  - M: an access matrix
  - label functions
    - $f_S: S \to L$ subject labels
    - $f_O: O \to L$ object labels
    - $f_C: S \to L$ current subject labels
  - object hierarchies are omitted

The Bell-LaPadula Model of Computer Systems

- Systems change states by handling requests
  - get/release access (change b)
  - change object level, current subject level (f_O,f_C)
  - give/rescind access permissions (M)

- Decisions to requests are
  - yes, no

End of Lecture 3

- Next lecture:
  - Partial orders, lattices, and security labels