Confidentiality Policies

CS461/ECE422 Computer Security I

Fall 2010

Reading

- Chapter 5 in CS
- Bell-LaPadula and McLean papers linked on class web site if you are interested in the proofs

Outline

- Overview
  - Mandatory versus discretionary controls
  - What is a confidentiality model
- Bell-LaPadula Model
  - General idea
  - Description of rules
- Tranquility
- Controversy
  - †-property
  - System Z

MAC vs DAC

- Discretionary Access Control (DAC)
  - Normal users can change access control state directly assuming they have appropriate permissions
  - Access control implemented in standard OS's, e.g., Unix, Linux, Windows
  - Access control is at the discretion of the user
- Mandatory Access Control (MAC)
  - Access decisions cannot be changed by normal rules
  - Generally enforced by system wide set of rules
  - Normal user cannot change access control schema
- "Strong" system security requires MAC
  - Normal users cannot be trusted

Confidentiality Policy

- Goal: prevent the unauthorized disclosure of information
  - Deals with information flow
  - Integrity incidental
- Multi-level security models are best-known examples
  - Bell-LaPadula Model basis for many, or most, of these

Bell-LaPadula Model, Step 1

- Security levels arranged in linear ordering
  - Top Secret: highest
  - Secret
  - Confidential
  - Unclassified: lowest
- Levels consist of *security clearance* $L(s)$
  - Objects have *security classification* $L(o)$

Example

security level	*subject*	*object*
Top Secret	Tamara	Personnel Files
Secret	Samuel	E-Mail Files
Confidential	Claire	Activity Logs
Unclassified	Ulaley	Telephone Lists

- Tamara can read all files
- Claire cannot read Personnel or E-Mail Files
- Ulaley can only read Telephone Lists

Reading Information

- Information flows *up*, not *down*
  - "Reads up" disallowed, "reads down" allowed
- Simple Security Condition (Step 1)
  - Subject s can read object o iff, $L(o) \le L(s)$ and s has permission to read o
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no reads up" rule

Writing Information

- Information flows up, not down
  - "Writes up" allowed, "writes down" disallowed
- *-Property (Step 1)
  - Subject s can write object o iff $L(s) \le L(o)$ and s has permission to write o
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no writes down" rule

Basic Security Theorem, Step 1

- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition (step 1), and the *-property (step 1), then every state of the system is secure
  - Proof: induct on the number of transitions
- Meaning of “secure” in axiomatic

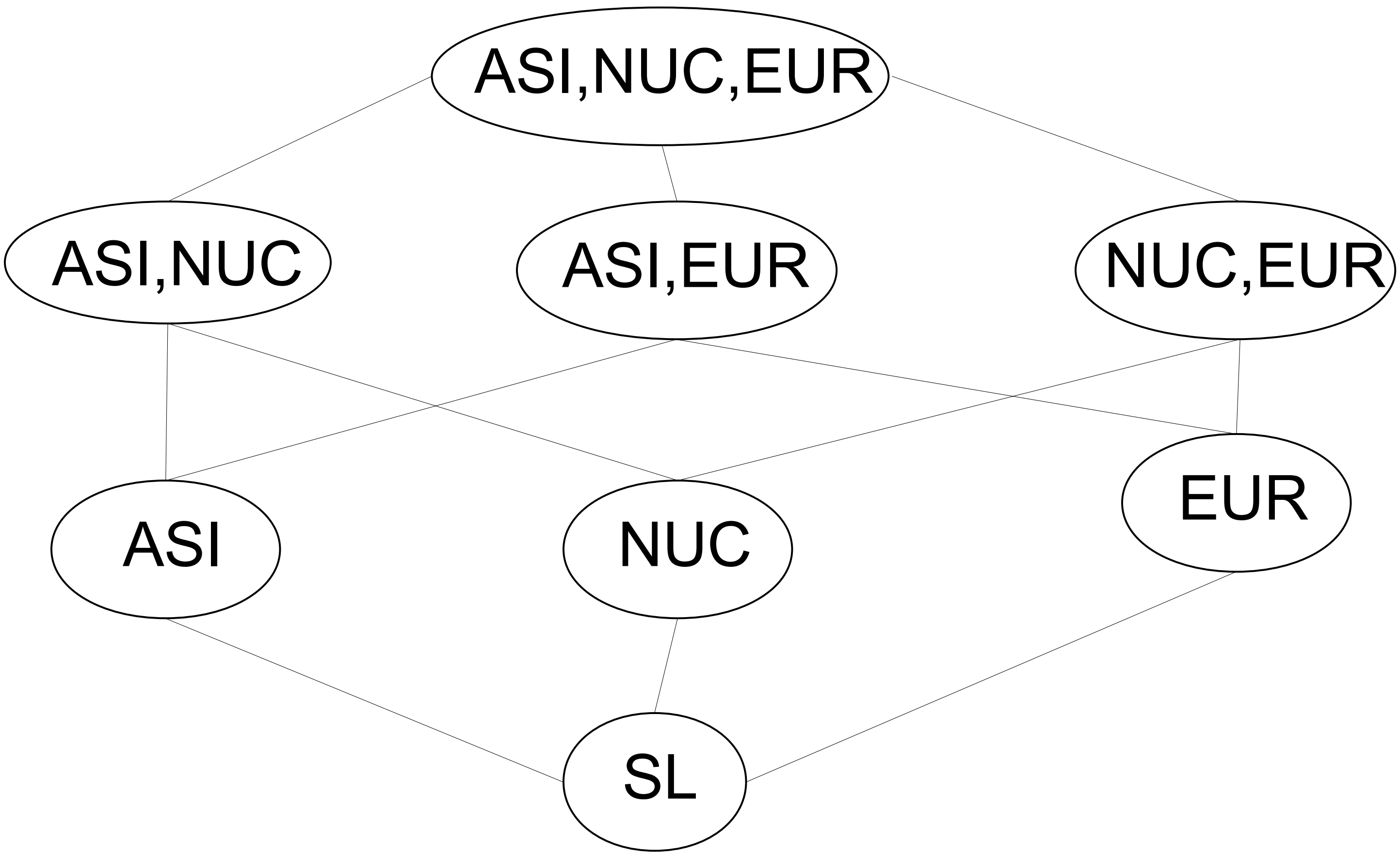
Bell-LaPadula Model, Step 2

- Expand notion of security level to include categories (also called compartments)
- Security level is (*clearance*, *category set*)
- Examples
  - (Top Secret, { NUC, EUR, ASI })
  - (Confidential, { EUR, ASI })
  - (Secret, { NUC, ASI })

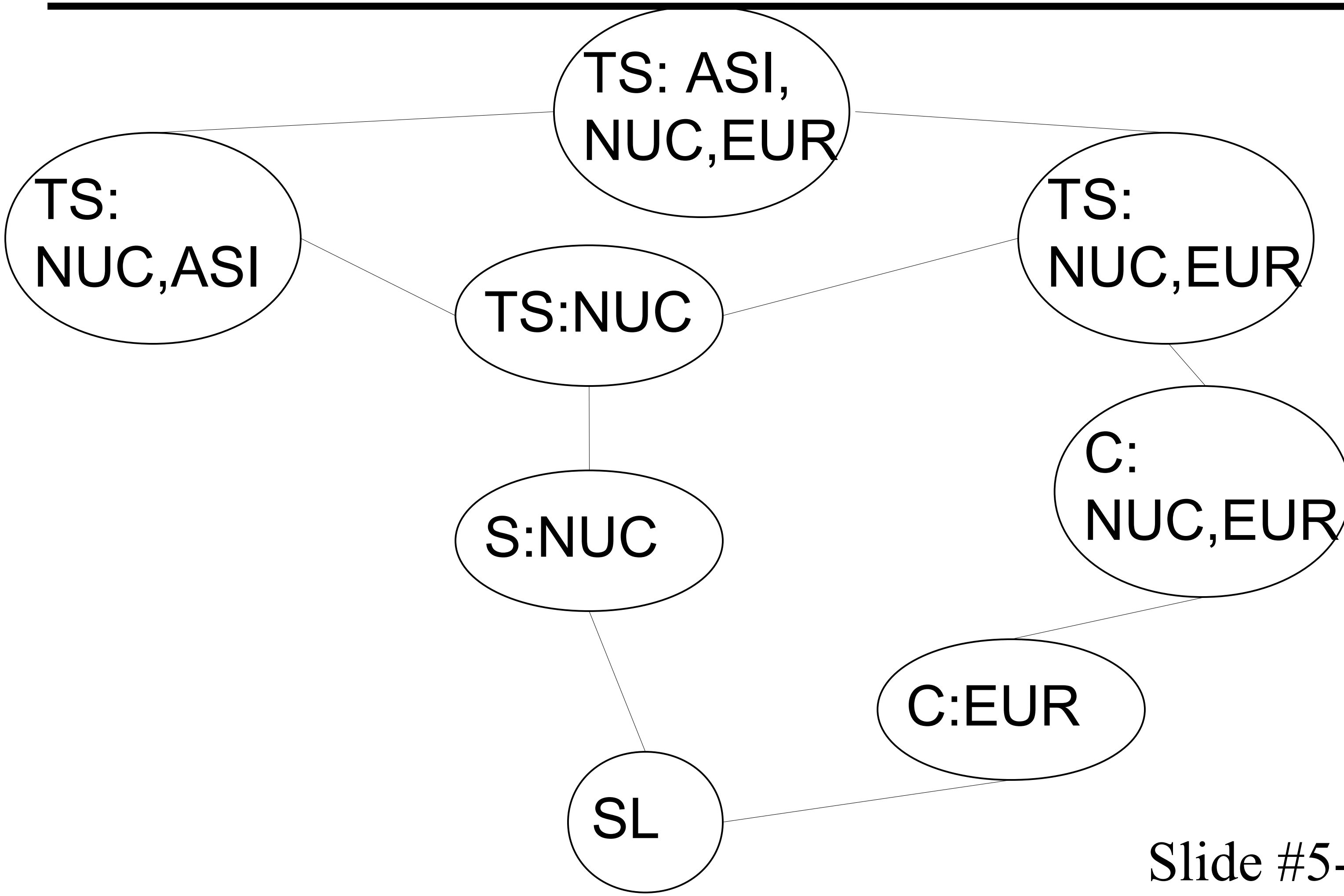
Levels and Lattices

- (A, C) *dom* (A', C') iff $A' \leq A$ and $C' \subseteq C$
- Examples
  - (Top Secret, {NUC, ASI}) *dom* (Secret, {NUC})
  - (Secret, {NUC, EUR}) *dom* (Confidential,{NUC, EUR})
  - (Top Secret, {NUC}) ¬*dom* (Confidential, {EUR})
  - (Secret, {NUC}) ¬*dom* (Confidential,{NUC, EUR})
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, *dom* form lattice
  - *Partially ordered set*
  - *Any pair of elements*
    - *Has a greatest lower bound*
    - *Has a least upper bound*

Example Lattice



Subset Lattice



Levels and Ordering

- Security levels partially ordered
  - Any pair of security levels may (or may not) be related by *dom*
- "dominates" serves the role of "greater than" in step 1
  - "greater than" is a total ordering, though

Reading Information

- Information flows *up*, not *down*
  - "Reads up" disallowed, "reads down" allowed
- Simple Security Condition (Step 2)
  - Subject *s* can read object *o* iff $L(s)$ *dom* $L(o)$ and *s* has permission to read *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no reads up" rule

Writing Information

- Information flows up, not down
  - "Writes up" allowed, "writes down" disallowed
- *-Property (Step 2)
  - Subject *s* can write object *o* iff $L(o)$ *dom* $L(s)$ and *s* has permission to write *o*
    - Note: combines mandatory control (relationship of security levels) and discretionary control (the required permission)
  - Sometimes called "no writes down" rule

Basic Security Theorem, Step 2

- If a system is initially in a secure state, and every transition of the system satisfies the simple security condition (step 2), and the *-property (step 2), then every state of the system is secure
  - Proof: induct on the number of transitions
  - In actual Basic Security Theorem, discretionary access control treated as third property, and simple security property and *-property phrased to eliminate discretionary part of the definitions — but simpler to express the way done here.

Problem

- Colonel has (Secret, {NUC, EUR}) clearance
- Major has (Secret, {EUR}) clearance
- Can Major write data that Colonel can read?
- Can Major read data that Colonel wrote?

Solution

- Define maximum, current levels for subjects
  - *maxlevel*(*s*) *dom curlevel*(*s*)
- Example
  - Treat Major as an object (Colonel is writing to him/her)
  - Colonel has *maxlevel* (Secret, { NUC, EUR })
  - Colonel sets *curlevel* to (Secret, { EUR })
  - Now *L*(Major) *dom curlevel*(Colonel)
    - Colonel can write to Major without violating “no writes down”
  - Does *L*(*s*) mean *curlevel*(*s*) or *maxlevel*(*s*)?
    - Formally, we need a more precise notation

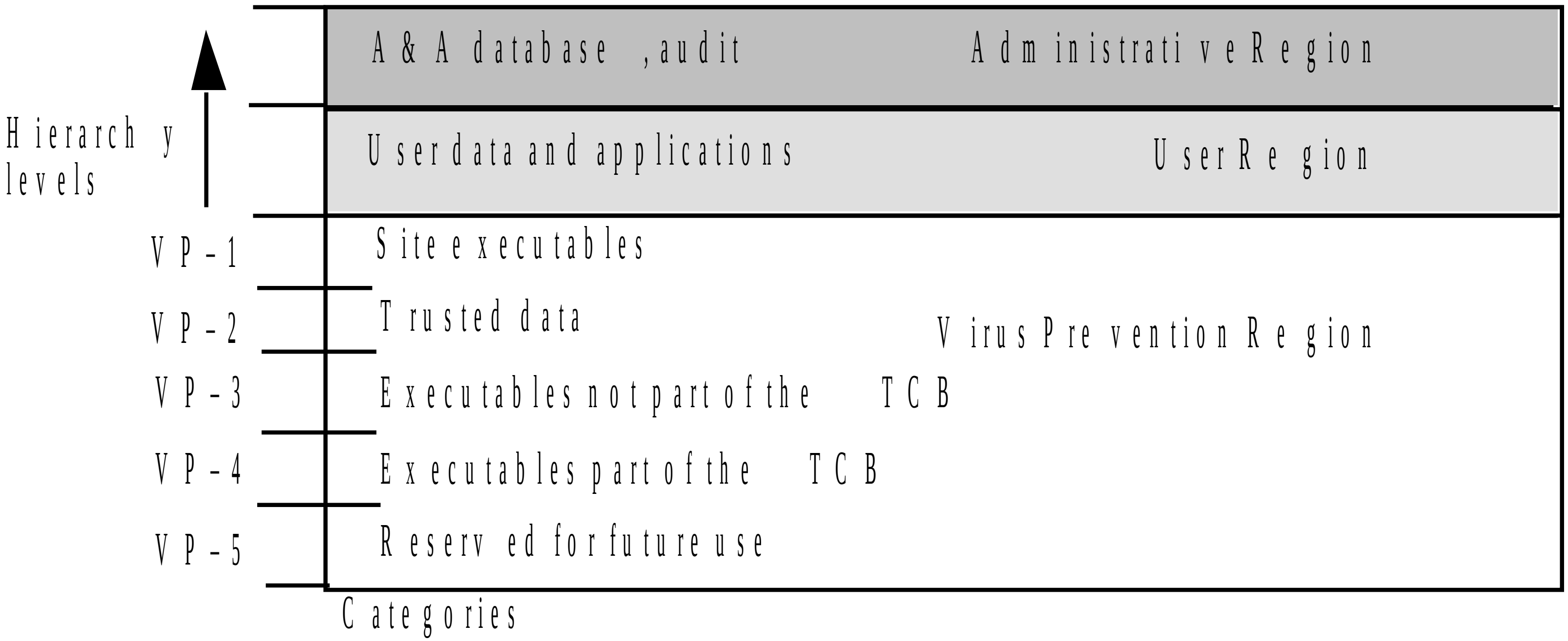
Adjustments to “write up”

- General write permission is both read and write
  - So both simple security condition and *-property apply
  - S dom O and O dom S means S=O
- BLP discuss **append** as a “pure” write so writeup still applies

DG/UX System

- Provides mandatory access controls
  - MAC label identifies security level
  - Default labels, but can define others
- Initially
  - Subjects assigned MAC label of parent
    - Initial label assigned to user, kept in Authorization and Authentication database
  - Object assigned label at creation
    - Explicit labels stored as part of attributes
    - Implicit labels determined from parent directory

MAC Regions



IMPL_HI is "maximum" (least upper bound) of all levels
IMPL_LO is "minimum" (greatest lower bound) of all levels

Directory Problem

- Process *p* at MAC_A tries to create file */tmp/x*
- */tmp/x* exists but has MAC label MAC_B
  - Assume MAC_B ¬ dom MAC_A
- Create fails
  - Now *p* knows a file named *x* with a higher label exists
- Fix: only programs with same MAC label as directory can create files in the directory
  - Now compilation won't work, mail can't be delivered

Multilevel Directory

- Directory with a set of subdirectories, one per label
  - Not normally visible to user
  - p creating */tmp/x* actually creates */tmp/d/x* where *d* is directory corresponding to MAC_A
  - All *p*'s references to */tmp* go to */tmp/d*
- *p* cd's to */tmp/a*, then to ..
  - System call stat(".", &buf) returns inode number of real directory
  - System call dg_stat(".", &buf) returns inode of */tmp*

Object Labels

- Requirement: every file system object must have MAC label

1. Roots of file systems have explicit MAC labels
   - If mounted file system has no label, it gets label of mount point
1. Object with implicit MAC label inherits label of parent

Object Labels

- Problem: object has two names
  - */x/y/z*, */a/b/c* refer to same object
  - *y* has explicit label IMPL_HI
  - *b* has explicit label IMPL_B
- Case 1: hard link created while file system on DG/UX system, so …

1. Creating hard link requires explicit label
   - If implicit, label made explicit
   - Moving a file makes label explicit

Object Labels

- Case 2: hard link exists when file system mounted
  - No objects on paths have explicit labels: paths have same *implicit* labels
  - An object on path acquires an explicit label: implicit label of child must be preserved

  so …
- Change to directory label makes child labels explicit *before* the change

Object Labels

- Symbolic links are files, and treated as such, so …

1. When resolving symbolic link, label of object is label of target of the link
   - System needs access to the symbolic link itself

Using MAC Labels

- Simple security condition implemented
- *-property not fully implemented
  - Process MAC must equal object MAC
  - Writing allowed only at same security level
- Overly restrictive in practice

MAC Tuples

- Up to 3 MAC ranges (one per region)
- MAC range is a set of labels with upper, lower bound
  - Upper bound must dominate lower bound of range
- Examples
  1. [(Secret, {NUC}), (Top Secret, {NUC})]
  - [(Secret, ∅), (Top Secret, {NUC, EUR, ASI})]
  1. [(Confidential, {ASI}), (Secret, {NUC, ASI})]

MAC Ranges

1. [(Secret, {NUC}), (Top Secret, {NUC})]
- [(Secret, $\emptyset$), (Top Secret, {NUC, EUR, ASI})]
1. [(Confidential, {ASI}), (Secret, {NUC, ASI})]
- (Top Secret, {NUC}) in ranges 1, 2
- (Secret, {NUC, ASI}) in ranges 2, 3
- [(Secret, {ASI}), (Top Secret, {EUR})] not valid range
  - as (Top Secret, {EUR}) $\neg$*dom* (Secret, {ASI})

Objects and Tuples

- Objects must have MAC labels
  - May also have MAC label
  - If both, tuple overrides label
- Example
  - Paper has MAC range:
    [(Secret, {EUR}), (Top Secret, {NUC, EUR})]

MAC Tuples

- Process can read object when:
  - Object MAC range (*lr*, *hr*); process MAC label *pl*
  - *pl dom hr*
    - Process MAC label grants read access to upper bound of range
- Example
  - Peter, with label (Secret, {EUR}), cannot read paper
    - (Secret, {EUR}) ¬ *dom* (Top Secret, {NUC, EUR})
  - Paul, with label (Top Secret, {NUC, EUR, ASI}) can read paper
    - (Top Secret, {NUC, EUR, ASI}) *dom* (Top Secret, {NUC, EUR})

MAC Tuples

- Process can write object when:
  - Object MAC range (*lr*, *hr*); process MAC label *pl*
  - $pl \in (lr, hr)$
    - Process MAC label grants write access to any label in range
- Example
  - Peter, with label (Secret, {EUR}), can write paper
    - (Top Secret, {NUC, EUR}) *dom* (Secret, {EUR}) and (Secret, {EUR}) *dom* (Secret, {EUR})
  - Paul, with label (Top Secret, {NUC, EUR, ASI}), cannot read paper
    - (Top Secret, {NUC, EUR}) ¬ *dom* (Top Secret, {NUC, EUR, ASI})

Formal Model

- S subjects, O objects, P rights
  - Defined rights: r read, a write, w read/write, e empty
- M set of possible access control matrices
  - That is, $m \in M$ iff $m \subseteq S \times O \times P$
- Let C be a set of clearances, and K a set of categories
  - $L = C \times K$ set of security levels

Security Level Assignments

- $F = \{ (f_s, f_o, f_c) \}$
- $f_s : S \rightarrow L$
  - $f_s(s)$ *maximum* security level of subject s
- $f_o : S \rightarrow L$
  - $f_o(o)$ security level of object o
- $f_c : S \rightarrow L$
  - $f_c(s)$ *current* security level of subject s

More Definitions

- Hierarchy functions $h: O \rightarrow P(O)$
- Requirements
  - $o_i \neq o_j \Rightarrow h(o_i) \cap h(o_j) = \varnothing$
  - There is no set $\{ o_1, \ldots, o_{k+1} \} \subseteq O$ such that, for $i = 1, \ldots, k$, $o_{i+1} \in h(o_i)$ and $o_{k+1} = o_1$.
- Defines a tree
  - Tree hierarchy; take $h(o)$ to be the set of children of o
  - No two objects have any common children (#1)
  - There are no loops (#2)

States and Requests

- V set of states
  - $v = (b, m, f, h) \in M \times M \times F \times (O \rightarrow P(O))$
    - *b mandatory rights*
    - *m discretionary rights*
    - *b* is like *m*, but excludes rights not allowed by *f*
- R set of requests for access
- D set of outcomes
  - <u>y</u> allowed, <u>n</u> not allowed, <u>i</u> illegal, <u>o</u> error

Actions

- W set of actions of the system
  - $W \subseteq R \times D \times V \times V$
  - (r,v) transitions to (d,v')

$$v \xrightarrow[(r,\, d,\, v,\, v') \in W]{r \text{ yields } d} v'$$

History

- X = set of sequences $(r_1, r_2, \ldots, r_t)$ of requests, $t \in N$ and $t > 1$
- Y = set of sequences $(d_1, d_2, \ldots, d_t)$ of decisions, $t \in N$ and $t > 1$
- Z = set of sequences $(v_0, v_1, \ldots, v_t)$ of states, $t \in N$
- Interpretation
  - At time $t \in N$, system is in state $z_{t-1} \in V$; request $x_t \in R$ causes system to make decision $y_t \in D$, transitioning the system into a (possibly new) state $z_t \in V$

History Continued

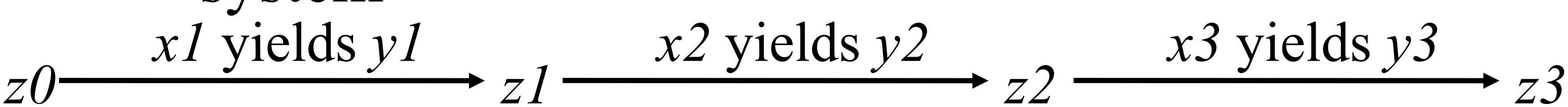
- System representation

  $\Sigma(R, D, W, z_0) \in X \times Y \times Z$

  – $(x, y, z) \in \Sigma(R, D, W, z_0)$ iff $(x_t, y_t, z_t, z_{t-1},) \in W$ for all t

  – (x, y, z) called an *appearance* of $\Sigma(R, D, W, z_0)$

  – Each z_t in an appearance (x, y, z) is a *state* of the system



$(x1, y1, z0, z1) \in W$

Rules

- A function $\rho : R \times V \rightarrow D \times V$ together with a start state determines a system

$$z0 \xrightarrow{x1 \text{ yields } \rho(x1,z0)} z1 \xrightarrow{x2 \text{ yields } \rho(x2,z1)} z2 \xrightarrow{x3 \text{ yields } \rho(x3,z2)} z3$$

Example

- $S = \{ s \}$, $O = \{ o \}$, $P = \{ \underline{\text{r}}, \underline{\text{w}} \}$
- $C = \{$ High, Low $\}$, $K = \{$ All $\}$
- For every $f \in F$, either $f_c(s) = ($ High, $\{$ All $\})$ or $f_c(s) = ($ Low, $\{$ All $\})$
- Initial State:
  - $b_1 = \{ (s, o, \underline{\text{r}}) \}$, $m_1 \in M$ gives s read access over o, and for $f_1 \in F$, $f_{c,1}(s) = ($High, $\{$All$\})$, $f_{o,1}(o) = ($Low, $\{$All$\})$
  - Call this state $v_0 = (b_1, m_1, f_1, h_1) \in V$.

First Transition

- Now suppose in state v_0: $S = \{ s, s' \}$
- Suppose $f_{c,1}(s') = (\text{Low}, \{\text{All}\})$
- $m_1 \in M$ gives s and s' read access over o
- As s' not written to o, $b_1 = \{ (s, o, \underline{\text{r}}) \}$
- $z_0 = v_0$; if s' requests r_1 to write to o:
  - System decides $d_1 = \underline{\text{y}}$
  - New state $v_1 = (b_2, m_1, f_1, h_1) \in V$
  - $b_2 = \{ (s, o, \underline{\text{r}}), (s', o, \underline{\text{w}}) \}$
  - Here, $x = (r_1)$, $y = (\underline{\text{y}})$, $z = (v_0, v_1)$

Second Transition

- Current state $v_1 = (b_2, m_1, f_1, h_1) \in V$
  - $b_2 = \{ (s, o, \underline{r}), (s', o, \underline{w}) \}$
  - $f_{c,1}(s) = (\text{High}, \{ \text{All} \}), f_{o,1}(o) = (\text{Low}, \{ \text{All} \})$
- s' requests r_2 to write to o:
  - System decides $d_2 = \underline{n}$ (as $f_{c,1}(s)$ dom $f_{o,1}(o)$)
  - New state $v_2 = (b_2, m_1, f_1, h_1) \in V$
  - $b_2 = \{ (s, o, \underline{r}), (s', o, \underline{w}) \}$
  - So, $x = (r_1, r_2)$, $y = (\underline{y}, \underline{n})$, $z = (v_0, v_1, v_2)$, where $v_2 = v_1$

Basic Security Theorem

- Define action, secure formally
  - Using a bit of foreshadowing for “secure”
- Restate properties formally
  - Simple security condition
  - *-property
  - Discretionary security property
- State conditions for properties to hold
- State Basic Security Theorem

Action

- A request and decision that causes the system to move from one state to another
  - Final state may be the same as initial state
- $(r, d, v, v') \in R \times D \times V \times V$ is an *action* of $\Sigma(R, D, W, z_0)$ iff there is an $(x, y, z) \in \Sigma(R, D, W, z_0)$ and a $t \in N$ such that $(r, d, v, v') = (x_t, y_t, z_{t-1}, z_t)$
  - Request r made when system in state v; decision d moves system into (possibly the same) state v'
  - Correspondence with (x_t, y_t, z_{t-1}, z_t) makes states, requests, part of a sequence

e empty
a write
r read
w read/write

le Security Condition

- $(s, o, p) \in S \times O \times P$ satisfies the *simple security condition relative to f* (written *ssc rel f*) iff one of the following holds:
  - p = e or p = a
  - p = r or p = w and $f_s(s)$ *dom* $f_o(o)$
- Holds vacuously if rights do not involve reading
- If all elements of *b* satisfy *ssc rel f*, then state satisfies simple security condition
- If all states satisfy simple security condition, system satisfies simple security condition

Necessary and Sufficient

- $\forall\ \Sigma(R, D, W, z_0)$ satisfies the simple security condition for a secure state z_0 iff every action $(r, d, (b, m, f, h), (b', m', f', h'))$ satisfies
  - Every $(s, o, p) \in b - b'$ satisfies *ssc rel f*
  - Every $(s, o, p) \in b'$ that does not satisfy *ssc rel f* is not in b
- Note: "secure" means z_0 satisfies *ssc rel f*
- First says every (s, o, p) added satisfies *ssc rel f*; second says any (s, o, p) in b' that does not satisfy *ssc rel f* is deleted

*-Property

- $b(s: p_1, \ldots, p_n)$ set of all objects that s has $p_1, \ldots, p_n$ access to
- State (b, m, f, h) satisfies the **-property* iff for each $s \in S$ the following hold:
  - $b(s: \underline{a}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{a}) [\, f_o(o) \; dom \; f_c(s) \,]\,]$
  - $b(s: \underline{w}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{w}) [\, f_o(o) = f_c(s) \,]\,]$
  - $b(s: \underline{r}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{r}) [\, f_c(s) \; dom \; f_o(o) \,]\,]$
- Idea: for writing, object dominates subject; for reading, subject dominates object

*-Property

- If all states satisfy simple security condition, system satisfies simple security condition
- If a subset S' of subjects satisfy *-property, then *-property satisfied relative to $S' \subseteq S$

Necessary and Sufficient

- ∀ $\Sigma(R, D, W, z_0)$ satisfies the *-property relative to $S' \subseteq S$ for any secure state z_0 iff every action $(r, d, (b, m, f, h), (b', m', f', h'))$ satisfies the following for every $s \in S'$
  - Every $(s, o, p) \in b - b'$ satisfies the *-property relative to S'
  - Every $(s, o, p) \in b'$ that does not satisfy the *-property relative to S' is not in b
- Note: "secure" means z_0 satisfies *-property relative to S'
- First says every (s, o, p) added satisfies the *-property relative to S'; second says any (s, o, p) in b' that does not satisfy the *-property relative to S' is deleted

Discretionary Security Property

- State (b, m, f, h) satisfies the *discretionary security property* iff, for each $(s, o, p) \in b$, then $p \in m[s, o]$
- Idea: if s can read o, then it must have rights to do so in the access control matrix m
- This is the discretionary access control part of the model
  - The other two properties are the mandatory access control parts of the model

Necessary and Sufficient

- $\forall\ \Sigma(R, D, W, z_0)$ satisfies the ds-property for any secure state z_0 iff every action $(r, d, (b, m, f, h), (b', m', f', h'))$ satisfies:
  - Every $(s, o, p) \in b - b'$ satisfies the ds-property
  - Every $(s, o, p) \in b'$ that does not satisfy the ds-property is not in b
- Note: "secure" means z_0 satisfies ds-property
- First says every (s, o, p) added satisfies the ds-property; second says any (s, o, p) in b' that does not satisfy the *-property is deleted

Secure

- A system is *secure* iff it satisfies:
  – Simple security condition
  – *-property
  – Discretionary security property
- A state meeting these three properties is also said to be secure

Basic Security Theorem

- $\forall\, \Sigma(R, D, W, z_0)$ is a secure system if z_0 is a secure state and W satisfies the conditions for the preceding three theorems
  - The theorems are on the slides titled "Necessary and Sufficient"

Example Instantiation: Multics

- 11 rules affect rights:
  - set to request, release access
  - set to give, remove access to different subject
  - set to create, reclassify objects
  - set to remove objects
  - set to change subject security level
- Set of "trusted" subjects $S_T \subseteq S$
  - *-property not enforced; subjects trusted not to violate
- $\forall$ $\Delta(\rho)$ domain of a rule ρ
  - determines if components of request are valid

Bell, LaPadula 75

get-read Rule

- Request $r = (get, s, o, \underline{\mathrm{r}})$
  - s gets (requests) the right to read o
- Rule is $\rho_1(r, v)$:

if $(r \neq \Delta(\rho_1))$ **then** $\rho_1(r, v) = (\underline{\mathrm{i}}, v)$;

else if $(f_s(s)\ dom\ f_o(o)$ **and** $[s \in S_T$ **or** $f_c(s)\ dom\ f_o(o)]$

and $r \in m[s, o])$

then $\rho_1(r, v) = (\underline{\mathrm{y}}, (b \cup \{ (s, o, \underline{\mathrm{r}}) \}, m, f, h))$;

else $\rho_1(r, v) = (\underline{\mathrm{n}}, v)$;

Security of Rule

- The get-read rule preserves the simple security condition, the *-property relative to S - S_T, and the ds-property
  - Proof
    - Let v satisfy all conditions. Let $\rho_1(r, v) = (d, v')$. If $v' = v$, result is trivial. Suppose $v' = (b' \cup \{ (s_2, o, \underline{r})\}, m, f, h)$ where $b' = b \cup \{ (s_2, o, \underline{r}) \}$.

Proof

- Consider the simple security condition.
  - From the choice of v', either $b' - b = \varnothing$ or $\{ (s_2, o, \underline{\text{r}}) \}$
  - If $b' - b = \varnothing$, then $\{ (s_2, o, \underline{\text{r}}) \} \in b$, so $v = v'$, proving that v' satisfies the simple security condition.
  - If $b' - b = \{ (s_2, o, \underline{\text{r}}) \}$, because the *get-read* rule requires that $f_c(s)$ *dom* $f_o(o)$, an earlier result says that v' satisfies the simple security condition.

Proof

- Consider the *-property relative to S - S_T.
  - Either $s_2 \in S_T$ or $f_c(s)$ *dom* $f_o(o)$ from the definition of *get-read*
  - If $s_2 \in S_T$, then there is nothing to prove.
  - If $f_c(s)$ *dom* $f_o(o)$, then condition 3 of the *-property is trivially satisfied.

Proof

- Consider the discretionary security property.
  - Conditions in the *get-read* rule require $\underline{r} \in m[s, o]$ and either $b' - b = \varnothing$ or $\{ (s_2, o, \underline{r}) \}$
  - If $b' - b = \varnothing$, then $\{ (s_2, o, \underline{r}) \} \in b$, so $v = v'$, proving that v' satisfies the simple security condition.
  - If $b' - b = \{ (s_2, o, \underline{r}) \}$, then $(s_2, o, \underline{r})$ is in m because that is a condition in the definition of $\rho_{1.}$

Principle of Tranquility

- Raising object's security level
  - Information once available to some subjects is no longer available
  - Usually assume information has already been accessed, so this does nothing
- Lowering object's security level
  - The *declassification problem*
  - Essentially, a "write down" violating *-property
  - Solution: define set of trusted subjects that *sanitize* or remove sensitive information before security level lowered

Types of Tranquility

- Strong Tranquility
  - The clearances of subjects, and the classifications of objects, do not change during the lifetime of the system
- Weak Tranquility
  - The clearances of subjects, and the classifications of objects change in accordance with a specified policy.

Example

- DG/UX System
  - Only a trusted user (security administrator) can lower object's security level
  - In general, process MAC labels cannot change
    - If a user wants a new MAC label, needs to initiate new process
    - Cumbersome, so user can be designated as able to change process MAC label within a specified range
- Other systems allow multiple labeled windows to address users operating a multiple levels

Controversy

- McLean:
  - “value of the BST is much overrated since there is a great deal more to security than it captures. Further, what is captured by the BST is so trivial that it is hard to imagine a realistic security model for which it does not hold.”
  - Basis: given assumptions known to be non-secure, BST can prove a non-secure system to be secure

McLean 85

†-Property

- State (b, m, f, h) satisfies the †-property iff for each $s \in S$ the following hold:
  - $b(s: \underline{a}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{a}) [f_c(s) \; dom \; f_o(o)]]$
  - $b(s: \underline{w}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{w}) [f_o(o) = f_c(s)]]$
  - $b(s: \underline{r}) \neq \varnothing \Rightarrow [\forall o \in b(s: \underline{r}) [f_c(s) \; dom \; f_o(o)]]$
- Idea: for writing, subject dominates object; for reading, subject also dominates object
- Differs from *-property in that the mandatory condition for writing is reversed
  - For *-property, it's object dominates subject

Analogues

The following two theorems can be proved

- $\Sigma(R, D, W, z_0)$ satisfies the †-property relative to $S' \subseteq S$ for any secure state z_0 iff for every action $(r, d, (b, m, f, h), (b', m', f', h'))$, W satisfies the following for every $s \in S'$
  – Every $(s, o, p) \in b - b'$ satisfies the †-property relative to S'
  – Every $(s, o, p) \in b'$ that does not satisfy the †-property relative to S' is not in b
- $\Sigma(R, D, W, z_0)$ is a secure system if z_0 is a secure state and W satisfies the conditions for the simple security condition, the †-property, and the ds-property.

Problem

- This system is *clearly* non-secure!
  - Information flows from higher to lower because of the †-property

System Z

- Only one transition rule
  - Get-read(s,o), if s dom o allow read and set all objects to system low
- This system meets BLP requirements for security given weak tranquility
  - Given secure initial state, each subsequent state is secure
- Points out the need to evaluate the transition rules

Discussion

- Role of Basic Security Theorem is to demonstrate that rules preserve security
- Key question: what is security?
  - Bell-LaPadula defines it in terms of 3 properties (simple security condition, *-property, discretionary security property)
  - Theorems are assertions about these properties
  - Rules describe changes to a *particular* system instantiating the model
  - Showing system is secure requires proving rules preserve these 3 properties

Key Points

- Confidentiality models restrict flow of information
- Bell-LaPadula models multilevel security
  – Cornerstone of much work in computer security
- Controversy over meaning of security
  – Different definitions produce different results