Policy Languages

- Express security policies in a precise way
- High-level languages
  - Policy constraints expressed abstractly
- Low-level languages
  - Policy constraints expressed in terms of program options, input, or specific characteristics of entities on system

High-Level Policy Languages

- Constraints expressed independent of enforcement mechanism
- Constraints restrict entities, actions
- Constraints expressed unambiguously
  - Requires a precise language, usually a mathematical, logical, or programming-like language

Example: Web Browser

- Goal: restrict actions of Java programs that are downloaded and executed under control of web browser
- Language specific to Java programs
- Expresses constraints as conditions restricting invocation of entities

Expressing Constraints

- Entities are classes, methods
  - Class: set of objects that an access constraint constrains
  - Method: set of ways an operation can be invoked
- Operations
  - Instantiation: *s* creates instance of class *c*: $s \dashv c$
  - Invocation: *s*1 executes object *s*2: $s1 \mapsto s2$
- Access constraints
  - **deny**(*s op x*) **when** *b*
  - While *b* is true, subject *s* cannot perform *op* on (subject or class) *x*; empty *s* means all subjects

DTEL

- Basis: access can be constrained by types
- Combines elements of low-level, high-level policy languages
  - Implementation-level constructs express constraints in terms of language types
  - Constructs do not express arguments or inputs to specific system commands

Example

- Goal: users cannot write to system binaries
- Subjects in administrative domain can
  - User must authenticate to enter that domain
- Subjects belong to domains:
  - *d_user* ordinary users
  - *d_admin* administrative users
  - *d_login* for login
  - *d_daemon* system daemons

Types

- Object types:
  - *t_sysbin* executable system files
  - *t_readable* readable files
  - *t_writable* writable files
  - *t_dte* data used by enforcement mechanisms
  - *t_generic* data generated from user processes
- For example, treat these as partitions
  - In practice, files can be readable and writable; ignore this for the example

Domain Representation

- Sequence
  - First component is list of programs that start in the domain
  - Other components describe rights subject in domain has over objects of a type

    (crwd->t_writable)

    means subject can create, read, write, and list (search) any object of type t_writable

d_daemon Domain

```
domain d_daemon = (/sbin/init),
      (crwd->t_writable),
      (rd->t_generic, t_readable, t_dte),
      (rxd->t_sysbin),
      (auto->d_login);
```

- Compromising subject in *d_daemon* domain does not enable attacker to alter system files
  - Subjects here have no write access
- When /sbin/init invokes login program, login program transitions into *d_login* domain

d_admin Domain

```
domain d_admin =
  (/usr/bin/sh, /usr/bin/csh, /usr/bin/ksh),
  (crwxd->t_generic),
  (crwxd->t_readable, t_writable, t_dte,
                                  t_sysbin),
  (sigtstp->d_daemon);
```

- *sigtstp* allows subjects to suspend processes in *d_daemon* domain
- Admin users use a standard command interpreter

d_user Domain

```
domain d_user =
       (/usr/bin/sh, /usr/bin/csh, /usr/bin/ksh),
       (crwxd->t_generic),
       (rxd->t_sysbin),
       (crwd->t_writable),
       (rd->t_readable, t_dte);
```

- No auto component as no user commands transition out of it
- Users cannot write to system binaries

d_login Domain

```
domain d_login =
  (/usr/bin/login),
  (crwd->t_writable),
  (rd->t_readable, t_generic, t_dte),
  setauth,
  (exec->d_user, d_admin);
```

- Cannot execute anything except the transition
  - Only /usr/bin/login in this domain
- *setauth* enables subject to change UID
- *exec* access to *d_user*, *d_admin* domains

Set Up

```
initial_domain = d_daemon;
```

– System starts in *d_daemon* domain

```
assign –r t_generic /;
assign –r t_writable /usr/var, /dev, /tmp;
assign –r t_readable /etc;
assign –r –s dte_t /dte;
assign –r –s t_sysbin /sbin, /bin,
                           /usr/bin, /usr/sbin;
```

– These assign initial types to objects
– –r recursively assigns type
– –s binds type to name of object (delete it, recreate it, still of given type)

Add Log Type

- Goal: users can't modify system logs; only subjects in *d_admin*, new *d_log* domains can

```
type t_readable, t_writable, t_sysbin,
                    t_dte, t_generic, t_log;
```

- New type *t_log*

```
domain d_log =
  (/usr/sbin/syslogd),
  (crwd->t_log),
  (rwd->t_writable),
  (rd->t_generic, t_readable);
```

- New domain *d_log*

Fix Domain and Set-Up

```
domain d_daemon = (/sbin/init),
  (crwd->t_writable),
  (rxd->t_readable),
  (rd->t_generic, t_dte, t_sysbin),
  (auto->d_login, d_log);
```

- Subject in *d_daemon* can invoke logging process
  - Can log, but not execute anything

```
assign -r t_log /usr/var/log;
assign t_writable /usr/var/log/wtmp,
  /usr/var/log/utmp;
```

- Set type of logs

Low-Level Policy Languages

- Set of inputs or arguments to commands
  - Check or set constraints on system
- Low level of abstraction
  - Need details of system, commands

Example: X Window System

- UNIX X11 Windowing System
- Access to X11 display controlled by list
  - List says what hosts allowed, disallowed access

```
xhost +groucho -chico
```

- Connections from host groucho allowed
- Connections from host chico not allowed

Example: tripwire

- File scanner that reports changes to file system and file attributes
  - *tw.config* describes what may change

```
/usr/mab/tripwire +gimnpsu012345678-a
```

    - Check everything but time of last access ("-a")
  - database holds previous values of attributes

Example Database Record

```
/usr/mab/tripwire/README 0 ..../. 100600 45763 1
  917 10 33242 .gtPvf .gtPvY .gtPvY 0
  .ZD4cc0Wr8i21ZKaI..LUOr3
  .0fwo5:hf4e4.8TAqd0V4ubv ?...... ...9b3
  1M4GX01xbGIX0oVuGo1h15z3
  ?:Y9jfa04rdzM1q:eqt1APgHk
  ?.Eb9yo.2zkEh1XKovX1:d0wF0kfAvC
  ?1M4GX01xbGIX2947jdyrior38h15z3 0
```

- file name, version, bitmask for attributes, mode, inode number, number of links, UID, GID, size, times of creation, last modification, last access, cryptographic checksums

Comments

- System administrators not expected to edit database to set attributes properly
- Checking for changes with tripwire is easy
  - Just run once to create the database, run again to check
- Checking for conformance to policy is harder
  - Need to either edit database file, or (better) set system up to conform to policy, then run tripwire to construct database

Example English Policy

- Computer security policy for academic institution
  - Institution has multiple campuses, administered from central office
  - Each campus has its own administration, and unique aspects and needs
- Authorized Use Policy
- Electronic Mail Policy

Authorized Use Policy

- Intended for one campus (Davis) only
- Goals of campus computing
  - Underlying intent
- Procedural enforcement mechanisms
  - Warnings
  - Denial of computer access
  - Disciplinary action up to and including expulsion
- Written informally, aimed at user community

Electronic Mail Policy

- Systemwide, not just one campus
- Three parts
  - Summary
  - Full policy
  - Interpretation at the campus

Summary

- Warns that electronic mail not private
  - Can be read during normal system administration
  - Can be forged, altered, and forwarded
- Unusual because the policy alerts users to the threats
  - Usually, policies say how to prevent problems, but do not define the threats

Summary

- What users should and should not do
  - Think before you send
  - Be courteous, respectful of others
  - Don't interfere with others' use of email
- Personal use okay, provided overhead minimal
- Who it applies to
  - Problem is UC is quasi-governmental, so is bound by rules that private companies may not be
  - Educational mission also affects application

Full Policy

- Context
  - Does not apply to Dept. of Energy labs run by the university
  - Does not apply to printed copies of email
    - Other policies apply here
- E-mail, infrastructure are university property
  - Principles of academic freedom, freedom of speech apply
  - Access without user's permission requires approval of vice chancellor of campus or vice president of UC
  - If infeasible, must get permission retroactively

Uses of E-mail

- Anonymity allowed
  - Provided it doesn't break laws or other policies
- Can't interfere with others' use of e-mail
  - No spam, letter bombs, e-mailed worms, *etc*.
- Personal e-mail allowed within limits
  - Cannot interfere with university business
  - Such e-mail may be a "university record" subject to disclosure

Security of E-mail

- University can read e-mail
  - Won't go out of its way to do so
  - Allowed for legitimate business purposes
  - Allowed to keep e-mail robust, reliable
- Archiving and retention allowed
  - May be able to recover e-mail from end system (backed up, for example)

Implementation

- Adds campus-specific requirements and procedures
  - Example: “incidental personal use” not allowed if it benefits a non-university organization
  - Allows implementation to take into account differences between campuses, such as self-governance by Academic Senate
- Procedures for inspecting, monitoring, disclosing e-mail contents
- Backups

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

Bell-LaPadula Model, Step 2

- Expand notion of security level to include categories
- Security level is (*clearance*, *category set*)
- Examples
  - (Top Secret, { Nuc, Eur, Asi })
  - (Confidential, { Eur, Asi })
  - (Secret, { Nuc, Asi })

Overview

- Lattices used to analyze Bell-LaPadula, Biba constructions
- Consists of a set and a relation
- Relation must partially order set
  - Partial ordering ≤ orders some, but not all, elements of set

Sets and Relations

- *S* set, *R*: $S \times S$ relation
  - If $a, b \in S$, and $(a, b) \in R$, write aRb
- Example
  - $I = \{ 1, 2, 3\}$; relation *R* is ≤
  - $R = \{ (1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3) \}$
  - So we write $1 \leq 2$ and $3 \leq 3$ but not $3 \leq 2$

Relation Properties

- Reflexive
  - For all $a \in S$, aRa
  - On I, $\leq$ is reflexive as $1 \leq 1$, $2 \leq 2$, $3 \leq 3$
- Antisymmetric
  - For all $a, b \in S$, $aRb \wedge bRa \Rightarrow a = b$
  - On I, $\leq$ is antisymmetric
- Transitive
  - For all $a, b, c \in S$, $aRb \wedge bRc \Rightarrow aRc$
  - On I, $\leq$ is transitive as $1 \leq 2$ and $2 \leq 3$ means $1 \leq 3$

Bigger Example

- C set of complex numbers
- $a \in C \Rightarrow a = a_R + a_I i$, a_R, a_I integers
- $a \leq_C b$ if, and only if, $a_R \leq b_R$ and $a_I \leq b_I$
- $a \leq_C b$ is reflexive, antisymmetric, transitive
  - As $\leq$ is over integers, and a_R , a_I are integers

Partial Ordering

- Relation *R* orders some members of set *S*
  - If all ordered, it's total ordering
- Example
  - ≤ on integers is total ordering
  - $\leq_C$ is partial ordering on *C* (because neither 3+5*i* $\leq_C$ 4+2*i* nor 4+2*i* $\leq_C$ 3+5*i* holds)

Upper Bounds

- For *a*, *b* ∈ *S*, if *u* in *S* with *aRu*, *bRu* exists, then *u* is upper bound
  - Least upper if there is no *t* ∈*S* such that *aRt*, *bRt*, and *tRu*
- Example
  - For 1 + 5*i*, 2 + 4*i* ∈ *C*, upper bounds include 2 + 5*i*, 3 + 8*i*, and 9 + 100*i*
  - Least upper bound of those is 2 + 5*i*

Lower Bounds

- For a, $b \in S$, if l in S with lRa, lRb exists, then l is lower bound
  - Greatest lower if there is no $t \in S$ such that tRa, tRb, and lRt
- Example
  - For $1 + 5i$, $2 + 4i \in C$, lower bounds include 0, $-1 + 2i$, $1 + 1i$, and $1 + 4i$
  - Greatest lower bound of those is $1 + 4i$

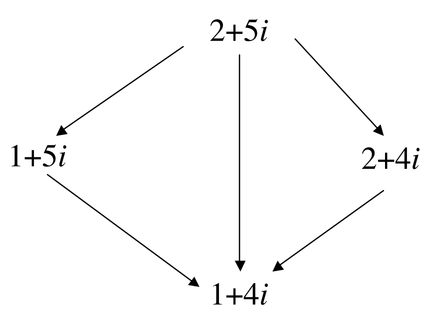
Lattices

- Set S, relation R
  - R is reflexive, antisymmetric, transitive on elements of S
  - For every s, $t \in S$, there exists a greatest lower bound under R
  - For every s, $t \in S$, there exists a least upper bound under R

Example

- C, $\leq_C$ form a lattice
  - As shown earlier, $\leq_C$ is reflexive, antisymmetric, and transitive
  - Least upper bound for a and b:
    - $c_R = \max(a_R, b_R)$, $c_I = \max(a_I, b_I)$; then $c = c_R + c_I i$
  - Greatest lower bound for a and b:
    - $c_R = \min(a_R, b_R)$, $c_I = \min(a_I, b_I)$; then $c = c_R + c_I i$

Picture



Arrows represent $\leq_C$

Levels and Lattices

- (A, C) *dom* (A', C') iff $A' \le A$ 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})
- Let C be set of classifications, K set of categories. Set of security levels $L = C \times K$, *dom* form lattice
  - $lub(L) = (max(A), C)$
  - $glb(L) = (min(A), \varnothing)$

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
  – Major can talk to colonel ("write up" or "read down")
  – Colonel cannot talk to major ("read up" or "write down")
- Clearly absurd!

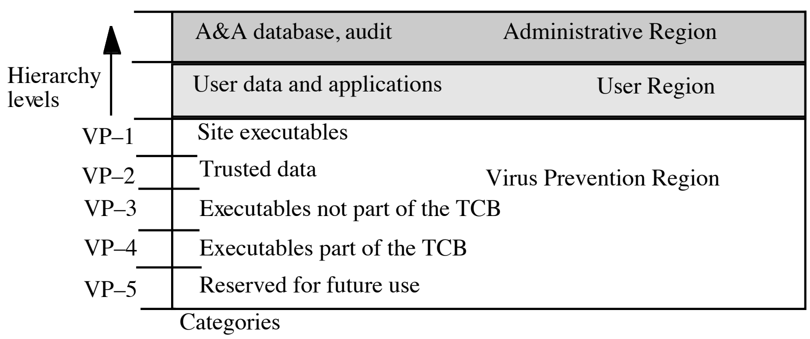
Solution

- 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

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 /tmpl