Mechanisms

- Entity or procedure that enforces some part of the security policy
  - Access controls (like bits to prevent someone from reading a homework file)
  - Disallowing people from bringing CDs and floppy disks into a computer facility to control what is placed on systems

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: $s_1 \mapsto s_2$
- 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

Sample Constraints

- Downloaded program cannot access password database file on UNIX system
- Program's class and methods for files:

```
class File {
  public file(String name);
  public String getfilename();
  public char read();
```

- Constraint:

```
deny( |-> file.read) when
   (file.getfilename() == "/etc/passwd")
```

Another Sample Constraint

- At most 100 network connections open
- *Socket* class defines network interface
  - *Network.numconns* method giving number of active network connections
- Constraint

```
deny( -| Socket) when
         (Network.numconns >= 100)
```

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
  - Exception: if it violates 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)

Chapter 5: Confidentiality Policies

- Overview
  - What is a confidentiality model
- Bell-LaPadula Model
  - General idea
  - Informal description of rules

Overview

- Goals of Confidentiality Model
- Bell-LaPadula Model
  - Informally
  - Example Instantiation

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 $L(o) \leq L(s)$ 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 })

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

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!

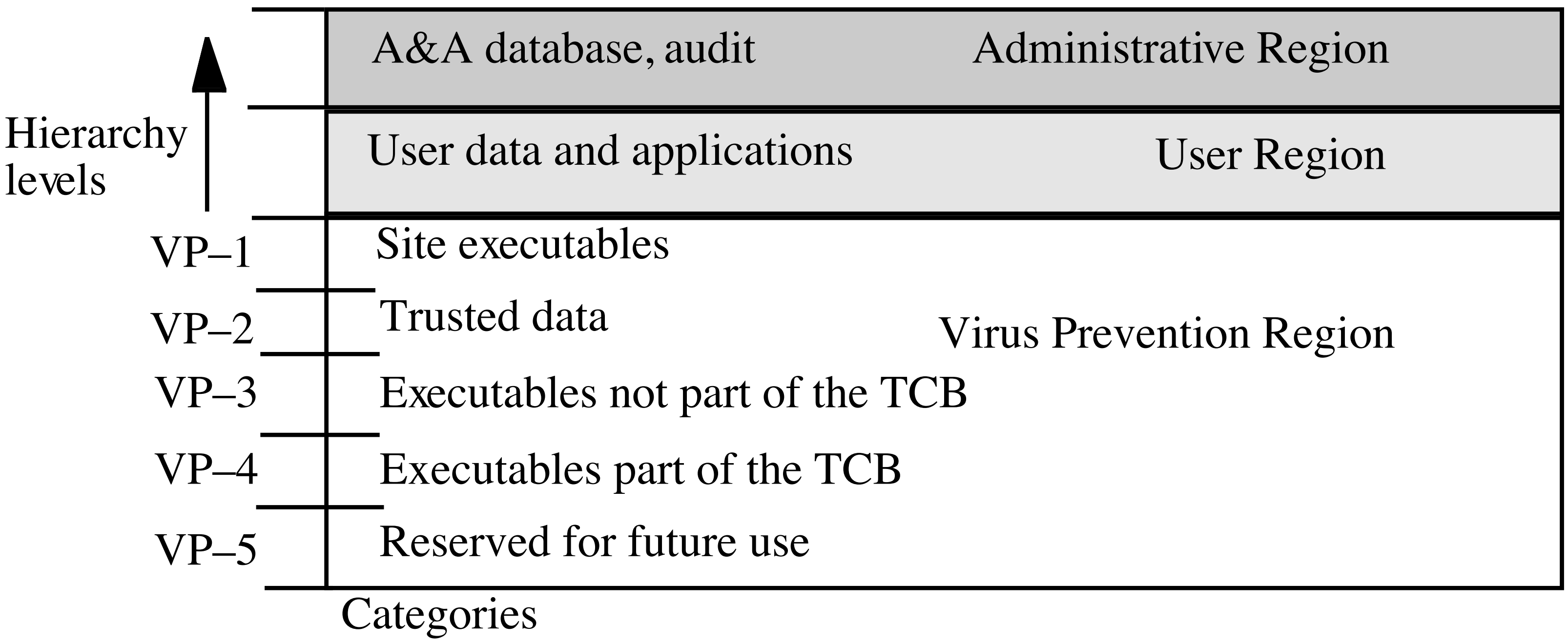
Solution

- 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 */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
2. 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 …

3. 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 …

4. Change to directory label makes child labels explicit *before* the change

Object Labels

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

5. 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})]
  2. [(Secret, ∅), (Top Secret, {NUC, EUR, ASI})]
  3. [(Confidential, {ASI}), (Secret, {NUC, ASI})]

MAC Ranges

1. [(Secret, {NUC}), (Top Secret, {NUC})]
2. [(Secret, ∅), (Top Secret, {NUC, EUR, ASI})]
3. [(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}) ¬*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
    - (Top Secret, {NUC, EUR}) *dom* (Secret, {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, ASI}) *dom* (Top Secret, {NUC, EUR})

Key Points

- Confidentiality models restrict flow of information
- Bell-LaPadula models multilevel security
  - Cornerstone of much work in computer security