UnitelmaSapienza
Università degli Studi di Roma

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Policies and Confidentiality

Prof. F. Parisi Presicce

Security Policies

- Overview
- The nature of policies
  - What they cover
  - Policy languages
- The nature of mechanisms
  - Types
  - Secure *vs.* precise
- What is a confidentiality model
- Bell-LaPadula Model
  - General idea
  - description of rules

Security Policy

- Policy partitions system states into:
  - Authorized (secure)
    - These are states the system can enter
  - Unauthorized (non-secure)
    - If the system enters any of these states, a breach of security has occurred
- Secure system
  - Starts in an authorized state
  - Never enters an unauthorized state

Question

- Policy disallows cheating
  - Includes copying homework, with or without permission
- CS class has students do homework on department computer
- Anne forgets to read-protect her homework file
- Bill notices this and copies it
- Who cheated?
  - Anne, Bill, or both?

Answer

‘Bill cheated

- Policy forbids copying homework assignment
- Bill did it
- System entered unauthorized state (Bill having a copy of Anne’s assignment)

Anne did not protect her homework

- Not required by security policy

She did not breach security

If policy said students had to read-protect homework files, then Anne did breach security

- Because she did not do this

Mechanisms

Entity or procedure that enforces some part of the security policy

- Access controls (set to prevent someone from reading a homework file)
- Disallowing people from bringing USB keys, CDs and floppy disks into a computer facility to control what is placed on systems

Policy Models

Abstract description of a policy or class of policies

Focus on points of interest in policies

- Confidentiality Policies
  - Prohibit direct or indirect information flow
  - Security levels in multilevel security models
- Integrity Policies
  - Restrict who/how data can be modified
  - Separation of duty in Clark-Wilson model
  - Conflict of interest in Chinese Wall model (both conf./int.)
- Availability Policies
  - describe what type/level of service must be provided

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 independently of enforcement mechanism

Constraints restrict actions and entities

Constraints expressed unambiguously

- Requires a precise language, usually a mathematical, logical, or programming-like language; English typically not precise enough

High Level Policy Languages

- **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 creation of classes and invocation of entities
- Independent of enforcement mechanism
- Entities are classes, methods
  - Class: set of objects that an access constraint constrains (e.g., file, socket)
  - Method: set of ways an operation can be invoked (e.g., file.read())
- Operations
  - Instantiation: *s* creates instance of class *c*: $s \dashv| c$
  - Invocation: s_1 executes object s_2: $s_1 |\rightarrow s_2$
- 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 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")
```

Program cannot open network connection when 100 connections already exist

- Constraint:

```
deny( |- socket) when (network.numconns >= 100)
```

Low-Level Policy Languages

Set of inputs or arguments to commands to check or set constraints on system

Low level of abstraction

- Need details of system, commands

UNIX-based X11 Windowing System

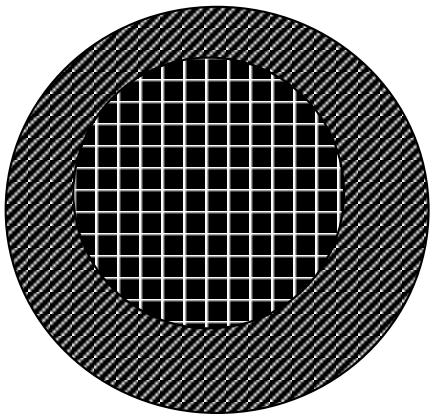
Access to X11 display controlled by list

- List says what hosts allowed or disallowed access

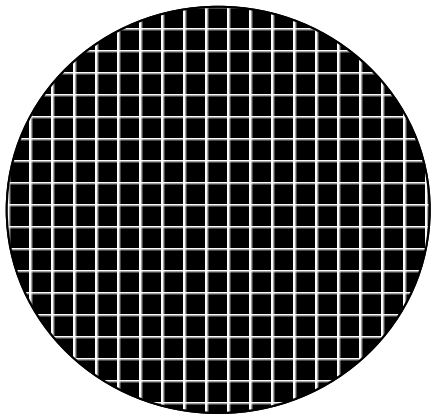
```
xhost +groucho -chico
```

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

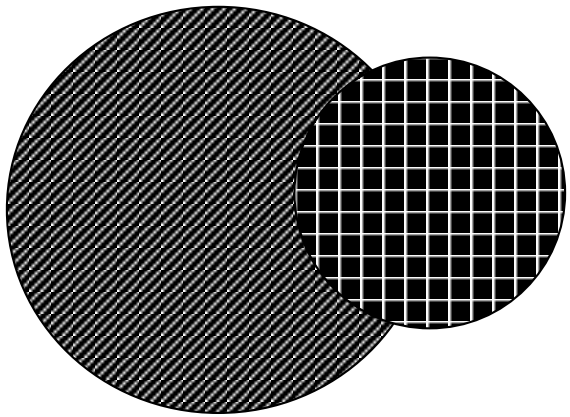
Types of Mechanisms



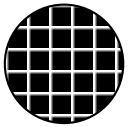
secure



precise



broad



set of reachable states

set of secure states

Secure, Precise Mechanisms

Can one devise a procedure for developing a mechanism that is precise?

Theorem

For any program *p* and security policy *c*, there exists a secure mechanism *m** such that, for all secure mechanisms *m* associated with *p* and *c*, $m \leq m^*$

- Minimizes number of denials of legitimate actions
- Maximally secure mechanism
  - But there is no effective procedure to construct it

Access Control Models

Often policies deal mostly with Access Control.

There two main categories:

- **Discretionary Access Control** (DAC, IBAC) where individual user sets access control mechanism to allow or deny access to an object; based on identity
  - Flexible and supported by most OS and DBMS
  - No Information flow and subject to Trojan Horse attacks
- **Mandatory Access Control** (MAC) where a system mechanism controls access to object, and individual cannot alter that access; based on rules
  - Often referred to as Multilevel security
  - MLS/DBMS databases satisfying multilevel security properties

Other Access Control Models

Other models

- **Chinese Wall Model** that combines aspects of DAC and MAC
- **Role Based Access Control** (RBAC) considered policy-neutral
- **Clark-Wilson Model** based on commercial policies and focused on information integrity
- **Information Flow models** generalizing ideas in MAC: what happens after access?

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 most of these
- Subjects and objects labeled with **security levels** that form a partial ordering.
- The system decides to grant or deny requests of access based on these levels
- No information flow allowed from ‘higher’ security levels down to ‘lower’ security level (confidentiality).

BLP model access classes

The classical notion of security level (top-secret, secret, confidential, unclassified) is extended to include categories (reflecting "need to know")

- An access class consists of two components, a security level (in totally ordered set) and a category set (dependent on application domain)
- **Security label** is pair (*clearance*, *category set*)
  - (*L*, *C*) *dominates* (*L´*, *C´*) iff *L´* ≤ *L* and *C´* ⊆ *C*
    - (Top Secret, {Aus,Asi}) *dom* (Secret, {Aus})
    - (Secret, {Aus, Eur}) *dom* (Confidential, {Aus,Eur})
    - (Top Secret, {Aus}) ¬*dom* (Confidential, {Eur})

" ≤ " is a total ordering on clearance, "dominates" is not, and forms a lattice

Constructing the State Set

All current access operations:

- an access operation is described by a triple $(s,o,a), s \in S, o \in O, a \in A$
  - e.g. (Alice, fun.com, read)
- The set of all current access operations is an element of $P(S \times O \times A)$
  - e.g. {(Alice, fun.com, read) , (Bob, fun.com, write), ...}

Constructing the State Set

Current assignment of security labels:

- maximal security label: f_S: $S \rightarrow L$ (L ... labels)
- current security label: f_C: $S \rightarrow L$
- classification: f_o: $O \rightarrow L$

The security label of a user is the user's clearance.

The security label of an object is its sensitivity level

Current security label allows subjects to be down-graded temporarily (more later).

$F \subseteq L^S \times L^S \times L^O$ is the set of security label assignments;
$f = (f_S, f_C, f_O)$ denotes an element of F.

Constructing the State Set

Current permissions:

- defined by the access control matrix M.
- M is the set of access control matrices.

The state set of BLP: $V = B \times M \times F$

- B is our shorthand for $P(S \times O \times A)$
- b denotes a set of current access operations
- a state is denoted by (b, M, f)

BLP model: Simple Security

Discretionary Security (ds)-Property : Access must be permitted by the access control matrix: if $(s,o,a) \in b$, then $a \in M_{so}$.

Simple Security (ss)-Property (no read-up): if $(s,o,a) \in b$ and access is in observe mode, then $f_S(s) \geq f_O(o)$.

The ss-property is a familiar policy for controlling access to classified paper documents.

On Subjects

- In the ss-property, subjects act as observers.
- In a computer system, subjects are *processes* and have no memory of their own.
- Subjects have access to memory objects.
- Subjects can act as channels by reading one memory object and transferring information to another memory object.
- In this way, data may be declassified improperly.

BLP model: Star Property

∗-Property (star property) (no write-down): if $(s,o,a) \in b$ and access is in alter mode, then $f_C(s) \leq f_O(o)$; also, if subject s has access to object o in alter mode, then $f_O(o') \leq f_O(o)$ for all objects o' accessed by s in observe mode.

(first version of BLP did not have ∗-property)

Mandatory BLP policies: ss-property and ∗-property.

No Write Down

- The $*$-property prevents high level subjects from sending legitimate messages to low level subjects (and to subjects not comparable).
- Two ways to escape from this restriction:
  - Temporarily downgrade high level subject; hence the current security level f_C; BLP subjects have **no** memory of their own!
  - Exempt trusted subjects from the $*$-property.
- Redefine the $*$-property and demand it only for subjects that are not trusted.

Basic Security Theorem

- A state is **secure**, if all current access tuples (s,o,a) are permitted by the ss-, *-, and ds-properties.
- A *state transition is secure* if it goes from a secure state to a secure state.

Basic Security Theorem: If the initial state of a system is secure and if all state transitions are secure, then the system will always be secure

Basic Security Theorem and Security

Construct system with operation *downgrade*:

- downgrades all subjects and objects to system low.
- enters all access rights in all positions of the access control matrix.

As a result, any state is secure in the BLP model.

Should such a system be regarded secure?

- McLean: no, everybody is allowed to do everything.
- Bell: yes, if *downgrade* was part of the system specification

(controversy in mid 80s)

DataGeneral B2 UniX System

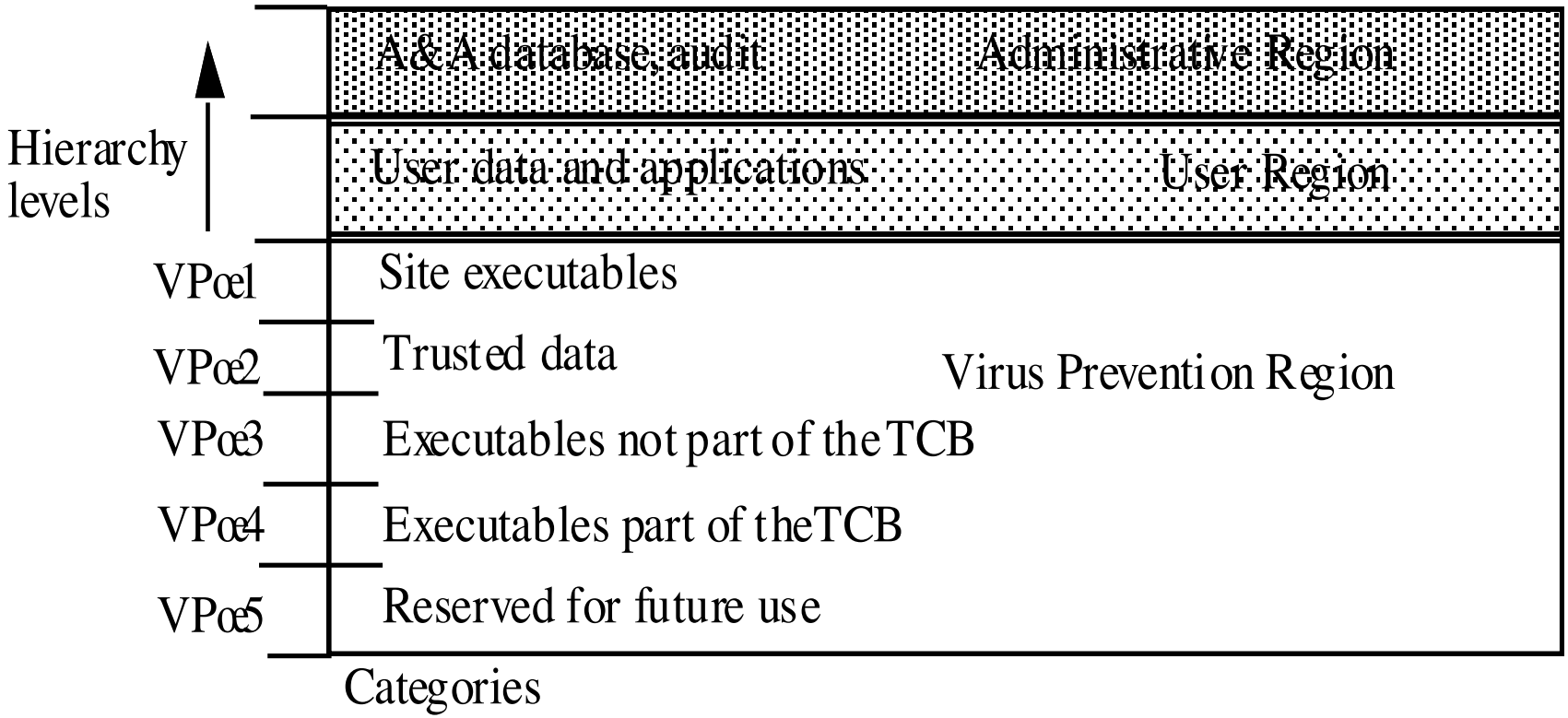
Provides mandatory access controls

- MAC label identifies security level
- Default labels discussed here, 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

The three MAC Regions



IMPL_HI is “maximum” (least upper bound) of all levels

IMPL_LO is “minimum” (greatest lower bound) of all levels

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

So … assign range

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 of range

1. [(Secret, {NUC}), (Top Secret, {NUC})]
2. [(Secret, ∅), (Top Secret, {NUC, EUR, ASI})]
3. [(Confidential, {ASI}), (Secret, {NUC, ASI})]
4. (Top Secret, {NUC}) in ranges 1, 2
5. (Secret, {NUC, ASI}) in ranges 2, 3
6. [(Secret, {ASI}), (Top Secret, {EUR})] **not** valid range

Objects and Tuples

Objects must have MAC labels

- May also have MAC tuple
- 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* ∈ (*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 write 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 last millennium