CIS 551 / TCOM 401

Computer and Network Security

Spring 2008
Lecture 6

Announcements

- Project 1 is due *this Friday* at 11:59
- Jianzhou's office hours
  - Weds. 1:30 - 2:30 in Levine 612

- Today: access control
  - Finish discussing windows
  - Capabilities
  - Multilevel security

Access Control Matrices

A[s][o]	Obj_1	Obj_2	…	Obj_N
$Subj_1$	{r,w,x}	{r,w}	…	{}
$Subj_2$	{w,x}	{}	…	
…	…	…	…	
$Subj_M$	{x}	{r,w,x}	…	{r,w,x}

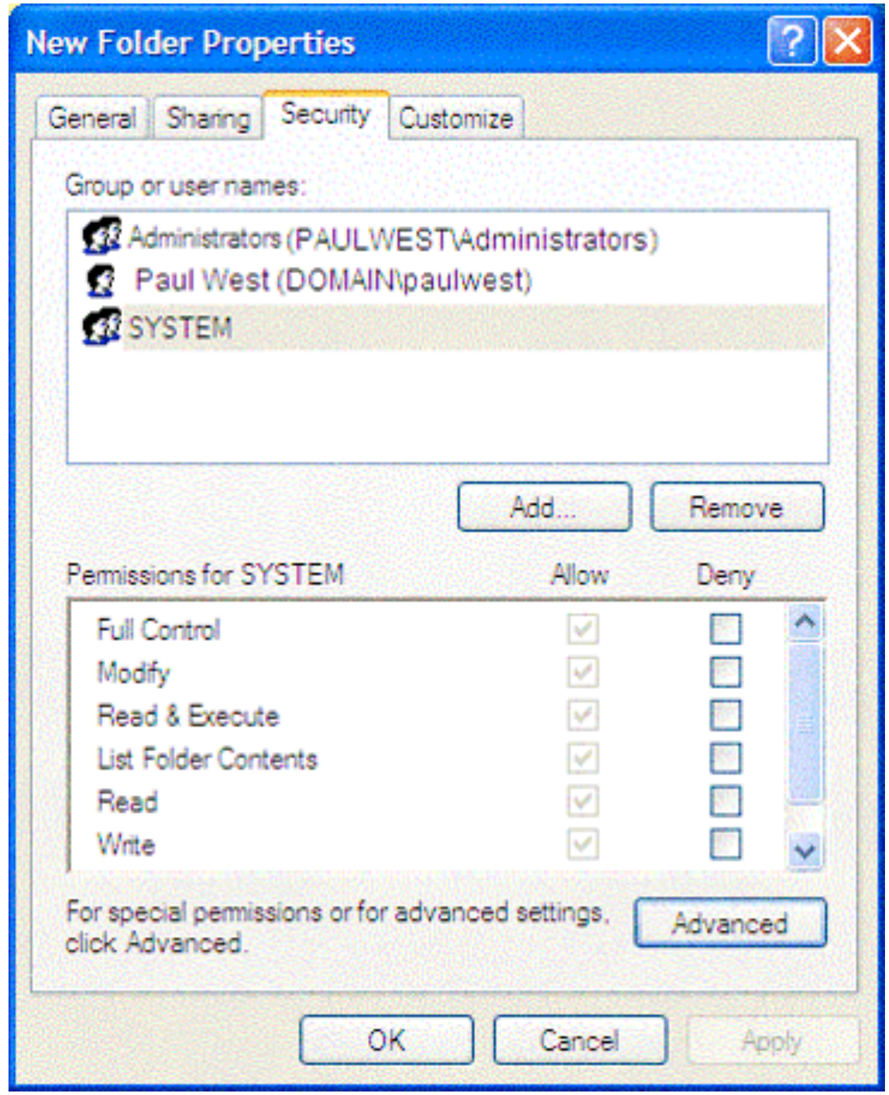
Each entry contains a set of rights.

Access control in Windows (NTFS)

- Some basic functionality similar to Unix
  - Specify access for groups and users
    - Read, modify, change owner, delete
  - ACLs used for fine grained control
- Some additional concepts
  - Tokens
  - Security attributes
- Generally
  - More flexibility than Unix
    - Can define new permissions
    - Can give some but not all administrator privileges

Sample permission options

- SID
  - "Security IDentifier"
  - Identity (like Unix UID)
    - SID revision number
    - 48-bit authority value
    - Globally unique
  - Describes users, groups, computers, domains, domain members



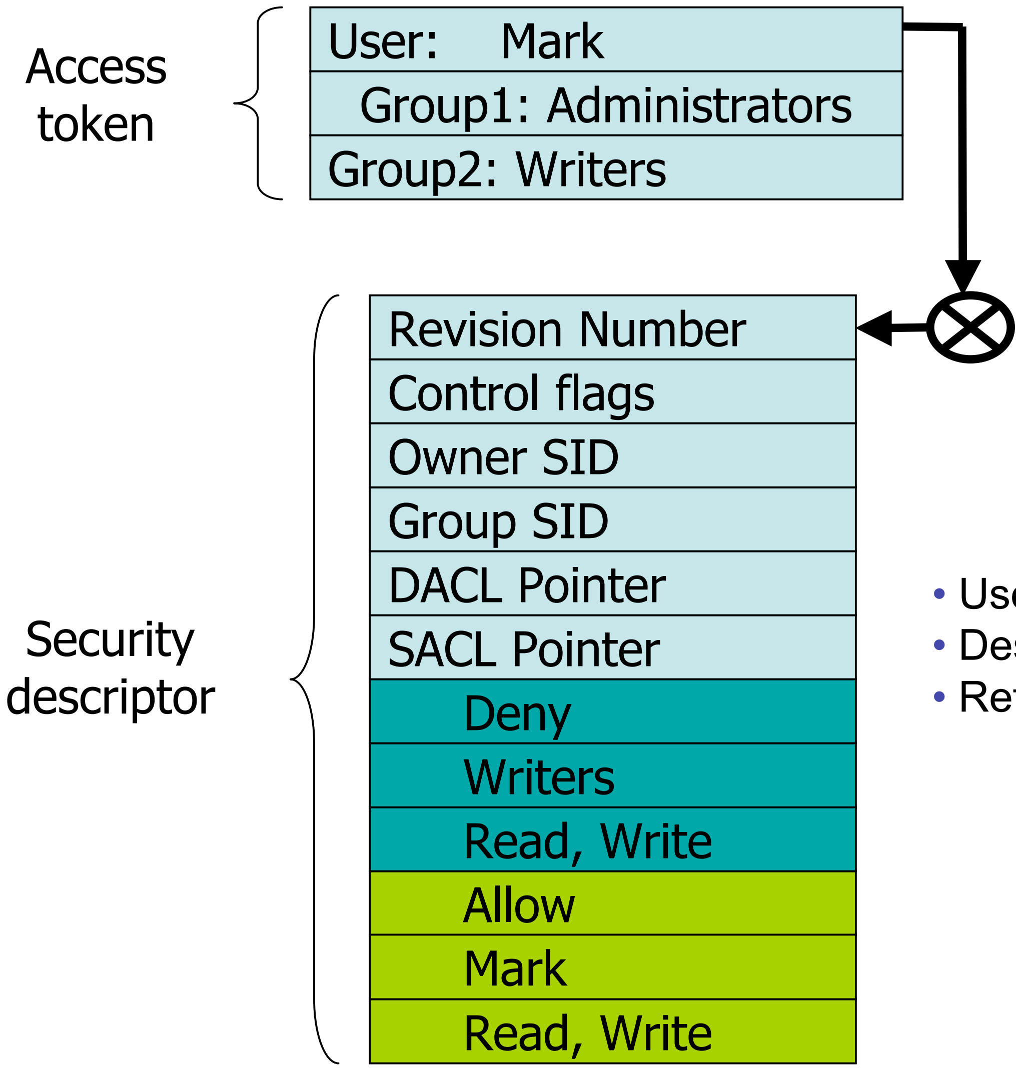
Security Descriptor

- Access Control List associated with an object
  - Specifies who can perform what actions on the object
- Several fields
  - Header
    - Descriptor revision number
    - Control flags, attributes of the descriptor
      - E.g., memory layout of the descriptor
  - SID of the object's owner
  - SID of the primary group of the object
  - Two attached optional lists:
    - Discretionary Access Control List (DACL)
      - Describes access policy
    - System Access Control List (SACL)
      - Describes audit/logging policy

Impersonation Tokens

- Windows equivalent of setuid
- Process uses security attributes of another
  - Client passes impersonation token to server
- Client specifies impersonation level of server
  - Anonymous
    - Token has no information about the client
  - Identification
    - server obtain the SIDs of client and client's privileges, but server cannot impersonate the client
  - Impersonation (= Anonymous + Identification)
    - server identify and impersonate the client
  - Delegation (= Impersonation + Authentication)
    - lets server impersonate client on local, remote system
- Tokens are a form of *capability*

Example access request



Access request: write

Action: denied

- User Mark requests write permission
- Descriptor denies permission to group
- Reference Monitor denies request

Windows Summary

- Good things
  - Very expressive
  - Don’t need full SYSTEM (e.g. root) privileges for many tasks
- Bad thing
  - More complex policies
    - Harder to implement: Larger TCB
    - Harder for users to understand
  - Wrong defaults
    - Users get administrator privileges by default
  - Historically, programs run with all privileges

Capabilities Lists

A[s][o]	Obj_1	Obj_2	…	Obj_N
$Subj_1$	{r,w,x}	{r,w}	…	{}
$Subj_2$	{w,x}	{}	…	{r}
…	…	…	…	…
$Subj_M$	{x}	{r,w,x}	…	{r,w,x}

For each subject, store a list of (Object x Rights) pairs.

Capabilities

- A capability is a (Object, Rights) pair
  - Used like a movie ticket e.g.:
    ("Cloverfield", {admit one, 7:00pm show})
- Should be unforgeable
  - Otherwise, subjects could get illegal access
- Authentication takes place when the capabilities are granted (not needed at use)
- Harder to do revocation (must find all tickets)
- Easy to audit a subject, hard to audit an object

Implementing Capabilities

- Must be able to name objects
- Unique identifiers
  - Must keep map of UIDs to objects
  - Must protect integrity of the map
  - Extra level of indirection to use the object
  - Generating UIDs can be difficult
- Pointers
  - Name changes when the object moves
  - Remote pointers in distributed setting
  - Aliasing possible

Unforgeability of Capabilities

- Special hardware: tagged words in memory
  - Can't copy/modify tagged words
- Store the capabilities in protected address space
- Could use static scoping mechanism of safe programming languages.
  - Java's "private" fields
- Could use cryptographic techniques
  - OS kernel could sign (Object, Rights) pairs using a private key
  - Any process can verify the capability
  - Example: Kerberos

Access Control

- *Discretionary*: The individual user may, at his own discretion, determine who is authorized to access the objects he creates.

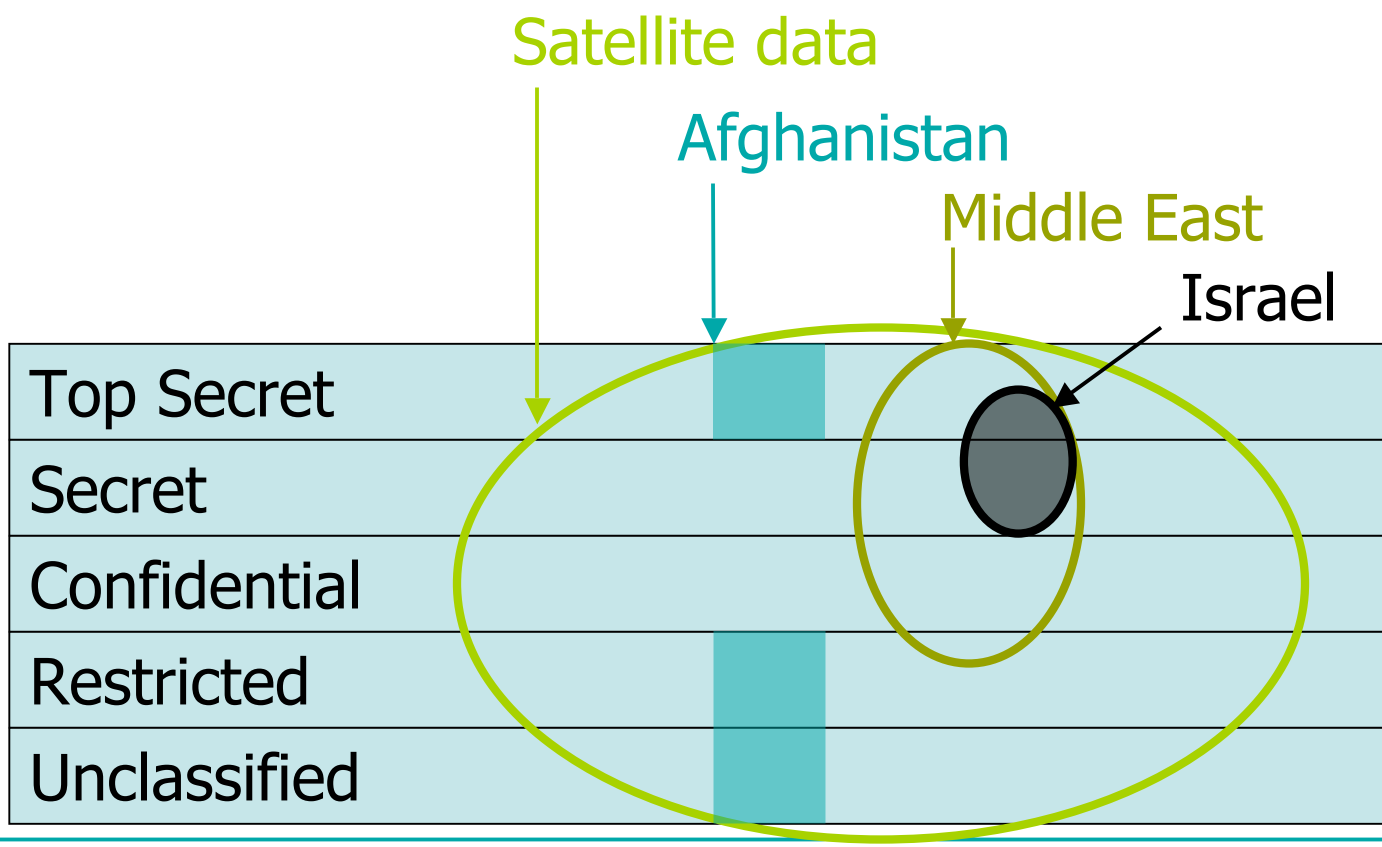
- *Mandatory*: The creator of an object does not necessarily have the ability to determine who has authorized access to it.
  - Typically policy is governed by some central authority
  - The policy on an object in the system depends on what object/information was used to create the object.

Multilevel Security

- Multiple levels of confidentiality or integrity ratings
- Military security policy
    - Classification involves sensitivity levels, compartments
    - Do not let classified information leak to unclassified files
- Group individuals and resources
  – Use some form of hierarchy to organize policy
- Trivial example: Public ≤ Secret
- *Information flow*
  – Regulate how information is used throughout entire system
  – A document generated from both Public and Secret information must be rated Secret.
  – Intuition: "Secret" information should not flow to "Public" locations.

Military security policy

- Sensitivity levels
- Compartments



Military security policy

- Classification of personnel and data
  - Class D = ⟨rank, compartment⟩
- Dominance relation
  - $D_1 \leq D_2$ iff $rank_1 \leq rank_2$
    and $compartment_1 \subseteq compartment_2$
  - Example: ⟨Restricted, Israel⟩ ≤ ⟨Secret, Middle East⟩
- Applies to
  - Subjects – users or processes: C(S) = "clearance of S"
  - Objects – documents or resources: C(O) = "classification of O"

Bell-LaPadula Confidentiality Model

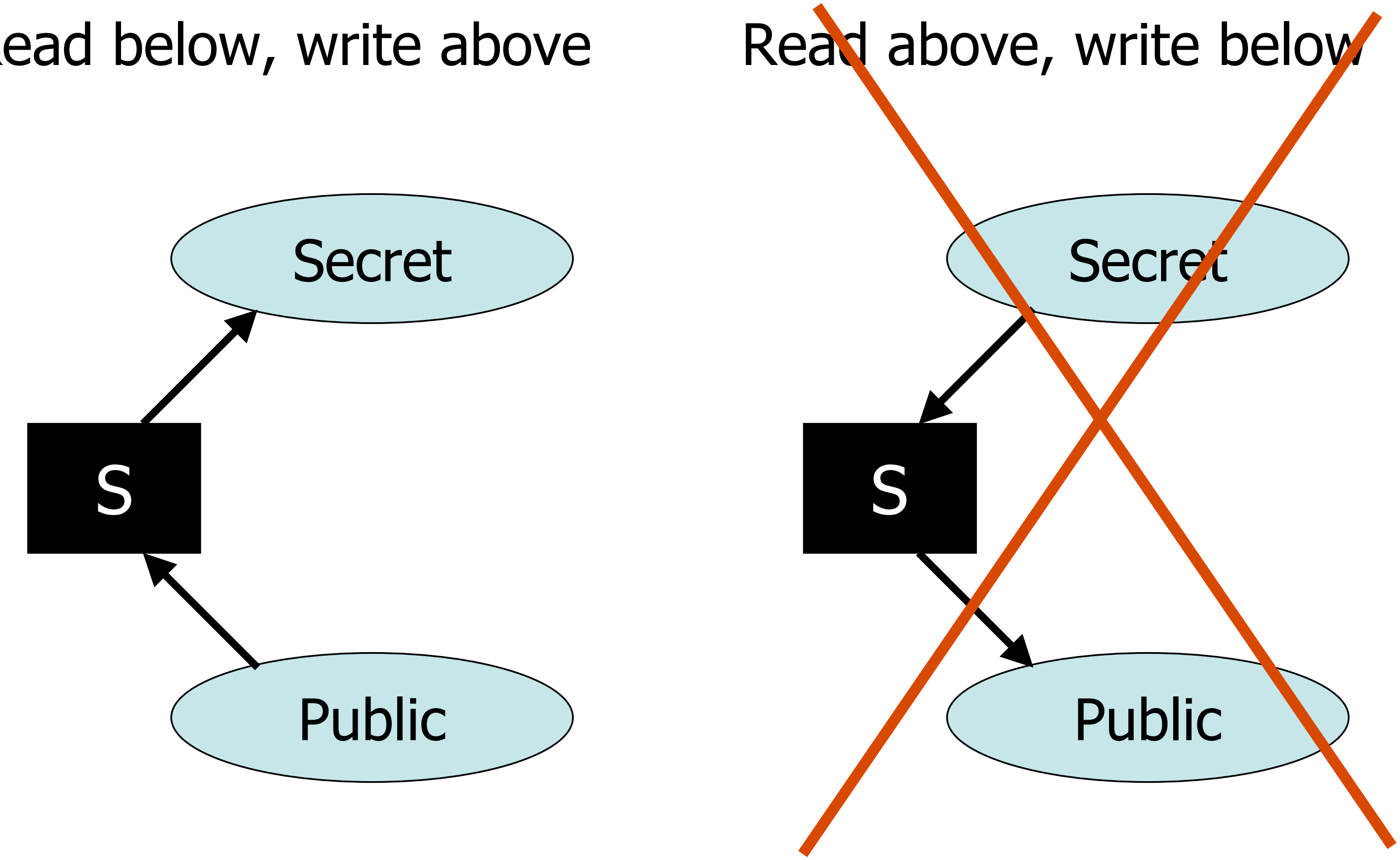
- “No read up, no write down.”
  - Subjects are assigned clearance levels drawn from the lattice of security labels.
    - C(S) = "clearance of the subject S"
  - A principal may read objects with lower (or equal) security label.
    - Read: $C(O) \leq C(S)$
  - A principal may write objects with higher (or equal) security label.
    - Write: $C(S) \leq C(O)$

- Example: A user with Secret clearance can:
  - Read objects with label Public and Secret
  - Write/create objects with label Secret

Picture: Confidentiality

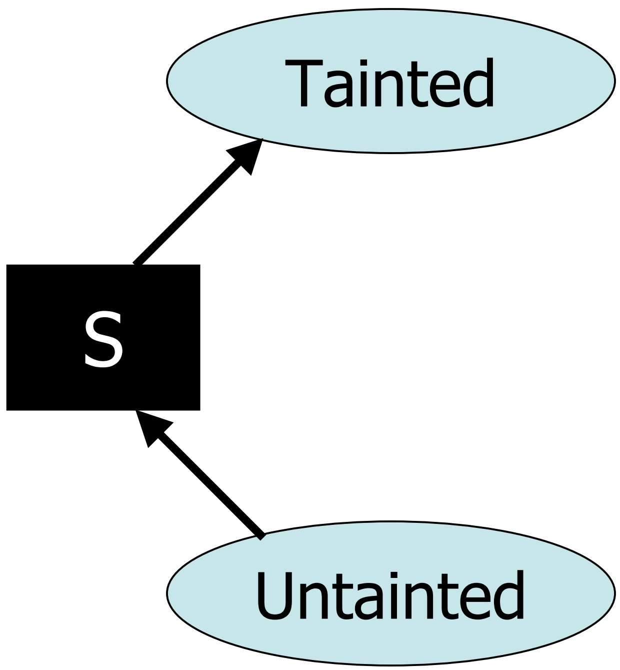
Read below, write above

~~Read above, write below~~

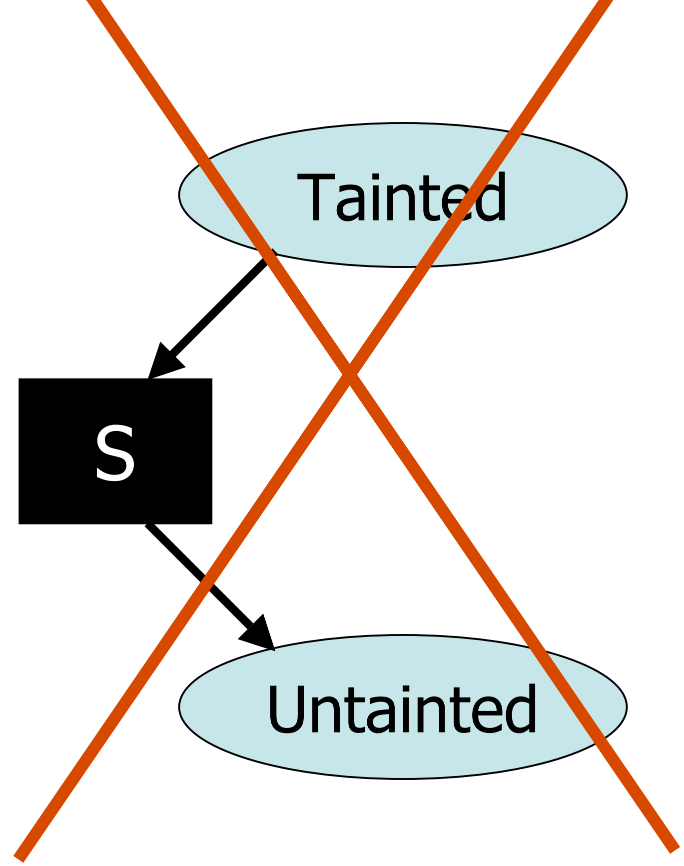


Picture: Integrity

Read below, write above



~~Read above, write below~~



Multilevel Security Policies

- In general, security levels form a "join semi-lattice"
  - There is an ordering $\leq$ on security levels
  - For any pair of labels L1 and L2 there is an "join" operation:

    L1 $\oplus$ L2 is a label in the lattice such that:
    (1) $L1 \leq L1 \oplus L2$ and $L2 \leq L1 \oplus L2$ "upper bound"
    (2) If $L1 \leq L3$ and $L2 \leq L3$ then $L1 \oplus L2 \leq L3$ "least bound"

- For example: Public $\oplus$ Secret = Secret
- Labeling rules:
  - Classification is a function C : Object $\rightarrow$ Lattice
  - If some object O is "created from" objects $O_1,\ldots,O_n$ then $C(O) = C(O_1) \oplus \ldots \oplus C(O_n)$

Implementing Multilevel Security

- Dynamic:
  - Tag all values in memory with their security level
  - Operations propagate security levels
  - Must be sure that tags can't be modified
  - Expensive, and approximate
- Classic result: Information-flow policies cannot be enforced purely by a reference monitor!
  - Problem arises from implicit flows
- Static:
  - Program analysis
  - May be more precise
  - May have less overhead

Information Flows through Software

Explicit Flows:

```
int{Secret} X = f();
int{Public} Y = 0;

Y = X;
```

Implicit Flows:

```
int{Secret} X = f();
int{Public} Y = 0;
int{Public} Z = 0;
int{Public} W = 0;

if (X > 0) then {
  Y = 1;
} else {
  Z = 1;
}
W = 3;
```

Perl's Solution (for Integrity)

- The problem: need to track the source of data
- Examples: Format string, SQL injection, etc.

```
$arg = shift;
system ("echo $arg");
```

- Give this program the argument `"; rm *"`
- Perl offers a *taint checking* mode
  - Tracks the source of data (trusted vs. tainted)
  - Ensure that tainted data is not used in system calls
  - Tainted data can be converted to trusted data by pattern matching
  - Doesn't check implicit flows

SELinux

- Security-enhanced Linux system (NSA)
  - Enforce separation of information based on confidentiality and integrity requirements
  - Mandatory access control incorporated into the major subsystems of the kernel
    - Limit tampering and bypassing of application security mechanisms
    - Confine damage caused by malicious applications

http://www.nsa.gov/selinux/

SELinux Security Policy Abstractions

- Security-Encanced Linux
  - Built by NSA
- Type enforcement
  - Each process has an associated domain
  - Each object has an associated type (label)
  - Configuration files specify
    - How domains are allowed to access types
    - Allowable interactions and transitions between domains
- Role-based access control
  - Each process has an associated role
    - Separate system and user processes
  - Configuration files specify
    - Set of domains that may be entered by each role

Two Other MAC Policies

- "Chinese Wall" policy: [Brewer & Nash '89]
  - Object labels are classified into "conflict classes"
  - If subject accesses one object with label L1 in a conflict class, all access to objects labeled with other labels in the conflict class are denied.
  - Policy changes dynamically

- "Separation of Duties":
  - Division of responsibilities among subjects
  - Example: Bank auditor cannot issue checks.

Covert Channels & Information Hiding

- A covert channel is a means by which two components of a system that are not permitted to communicate do so anyway by affecting a shared resource.

- Information hiding: Two components of the system that are permitted to communicate about one set of things, exchange information about disallowed topics by encoding contraband information in the legitimate traffic.

- Not that hard to leak a small amount of data
  - A 64 bit encryption key is not that hard to transmit
  - Even possible to encode relatively large amounts of data!

- Example channels / information hiding strategies
  - Program behavior
  - Adjust the formatting of output:
    use the “\t” character for “1” and 8 spaces for “0”
  - Vary timing behavior based on key
  - Use "low order" bits to send signals
  - Power consumption
  - Grabbing/releasing a lock on a shared resource