

# Computer Security

- Computer security deals with the control of the use and access to computer resources
  - ubiquitous presence of computers in every aspect of life implies use of reserved information is very common
  - not only misuse but also destruction and alteration of information should be prevented
- Computer security has become even more relevant in the era of e-commerce and on-line access to public services

# Security violations: classification

- Unauthorized information release
  - unauthorized access to data/applications
  - examples: password sniffing, illicit reserved data access
- Unauthorized information modification
  - unauthorized alteration of information
  - examples: web site defacing, hard disk erasing by virus
- Unauthorized denial of service
  - malicious overloading of server
  - examples: recent attacks to Cnn, Amazon, and others

# External vs. Internal Security

- External security = physical security
  - controlled access to computer premises, including terminals, console, etc.
  - involves guards, access keys, ...
- Internal security = logical security
  - controlled use of hardware peripherals and information stored on the system
  - involves user authentication, user permissions, etc.
- We are going to study internal security mechanisms

# Protection vs. security

- In general:
  - *policies* refer to what should be done
  - *mechanisms* refer to how it should be done
  - distinction between the two concept is desirable because enhances design flexibility
- In a operating system:
  - *protection* refers to the mechanism that enable selective control of user access to resources
  - *security* refers to the set of policies used to decide which users can have access to what resources

# Design principles for secure systems

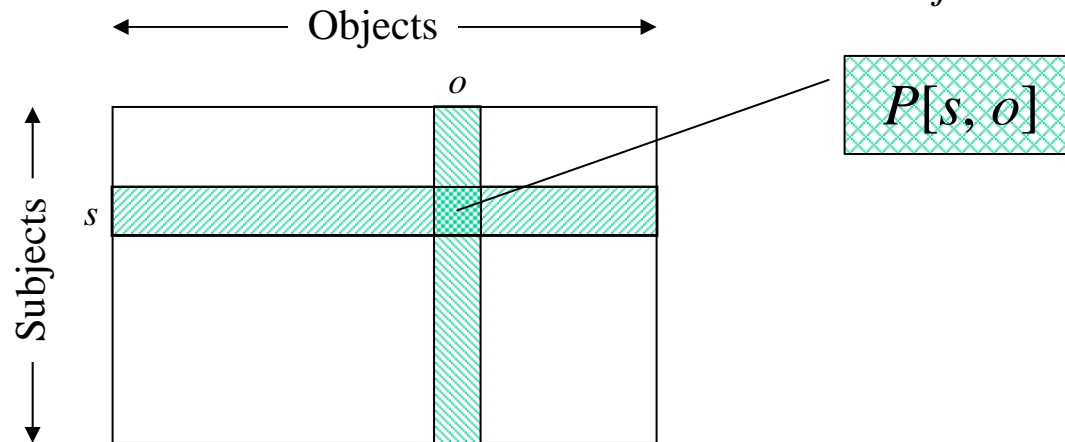
- Economy: low overhead
- Complete mediation: no holes in the protection
- Open design: secrets don't last long
- Least privilege: don't grant more rights than required
- acceptability: a hard to use mechanism won't be used
- Fail-Safe defaults: access should be denied in case of failure
- Least common mechanism: any coupling between users (i.e. shared variables) is a threat to their security

# Model of protection

- A model of protection abstracts the essential details of a protection mechanism
- We are going to study the *Access Matrix* model
  - first proposed by Lampson in 1971, the refined by others
  - very popular due to simplicity and elegance
  - common implementations:
    - capabilities
    - Access Control List

# Access Matrix Model

- Matrix used to define the set of access rights  $P[s_i, o_j]=\{r_1, r_2, \dots\}$  that every subject  $s_i$  has on every object  $o_j$



Example:

	$o_1$	$o_2$	$s_1$	$s_2$	$s_3$
$s_1$	<i>read, write</i>	<i>own, delete</i>	<i>own</i>	<i>sendmail</i>	<i>recmail</i>
$s_2$	<i>execute</i>	<i>copy</i>	<i>recmail</i>	<i>own</i>	<i>block, wakeup</i>
$s_3$	<i>own</i>	<i>read,write</i>	<i>sendmail</i>	<i>block, wakeup</i>	<i>own</i>

# Access Matrix: implementations

- Access matrix is likely to be very sparse
- Space-efficient implementations decompose the matrix in rows and/or columns:
  - *capabilities*: row-wise representation
  - *access control list*: column-wise representation
  - *lock-key*: combination of the above

# Capabilities

- This method associates a list of tuples  $(o, P[s, o])$  called *capabilities* to each subject  $s$ 
  - one tuple for every object  $o$  to which  $s$  is allowed access
  - capabilities cannot be modified by the subject
  - the list is a synthetic description of the row of the access matrix corresponding to  $s$

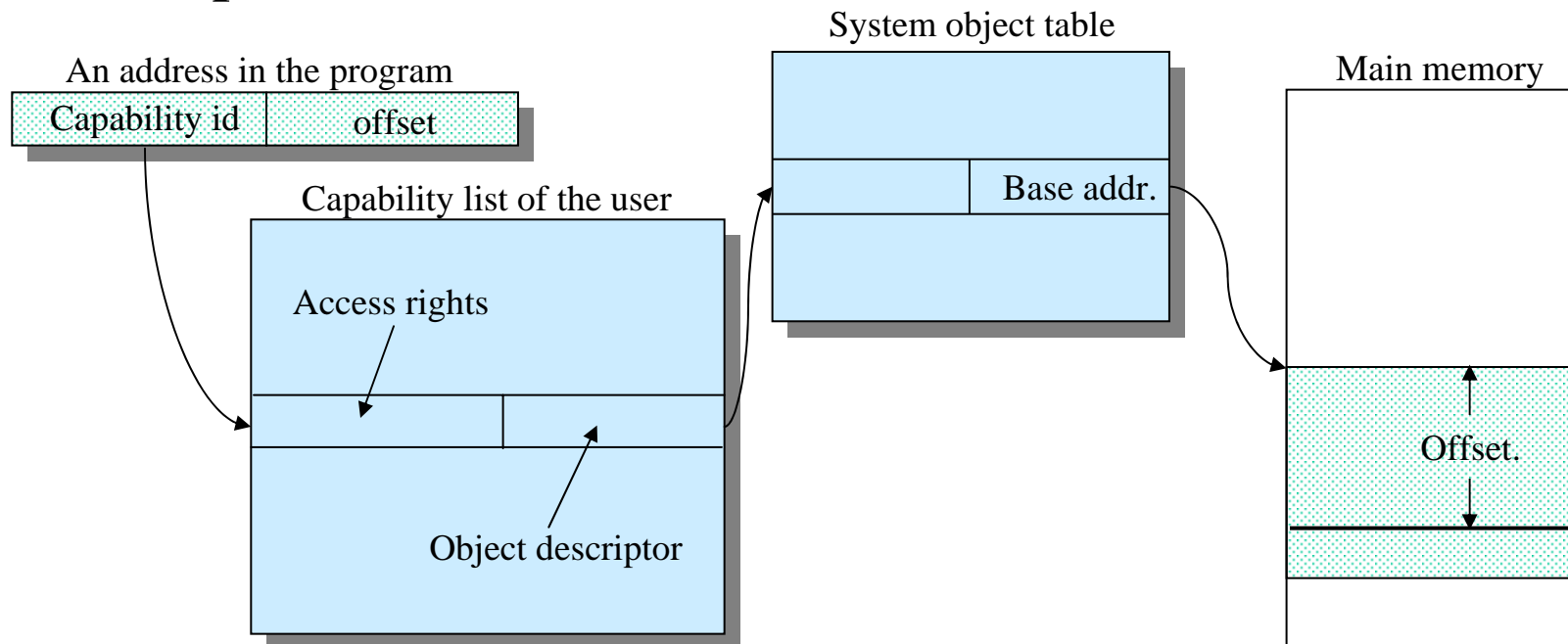
Example:

Object descriptor	Access rights (r, w, x, ...)
-------------------	------------------------------

- The capability concept can be extended and used as an addressing mechanism with useful properties

# Capability-based Addressing

- Basic idea: the object descriptor can be anything most convenient for the system, like the absolute address of the object or an index in a system table
- Example:



# Capability-based Addressing (2)

- Advantages:
  - relocability:
    - no change to the capabilities needed when relocating the object (only object table is modified)
  - easy sharing:
    - several program can share the same object (program, data) and with different names for the object
- Disadvantage:
  - added complication of mechanism to deny the user access to the capabilities
    - tagged approach, partitioned approach

# Capability-based protection: pros and cons

- Pros:
  - efficient: verifying access rights is fast
  - simple: the model itself is simple
  - flexible: a user can decide an arbitrary pattern of access rights for its data structures
- Cons:
  - if a subject gives a copy of a capability to another subject, the second subject can distribute copies without first subject's knowledge
  - difficult to find all those who have access to a certain object
  - garbage collection problem: when all capabilities referring to an object are deleted, the object cannot be accessed anymore

# The Access Control List method

- The ACL list method corresponds to the column-wise decomposition of the Access Matrix
  - each object  $o$  is associated with a list of pairs  $(s, P[s, o])$  for all subjects  $s$  that are allowed access to  $o$
  - such list is a short representation of  $o$ 's column in the matrix
- When a subject requests access to an object, the object's ACL is checked for permission
- Example:

Smith	read, write
Subraman.	read, execute
Maxwell	write
Qian	execute
Esposito	read, write

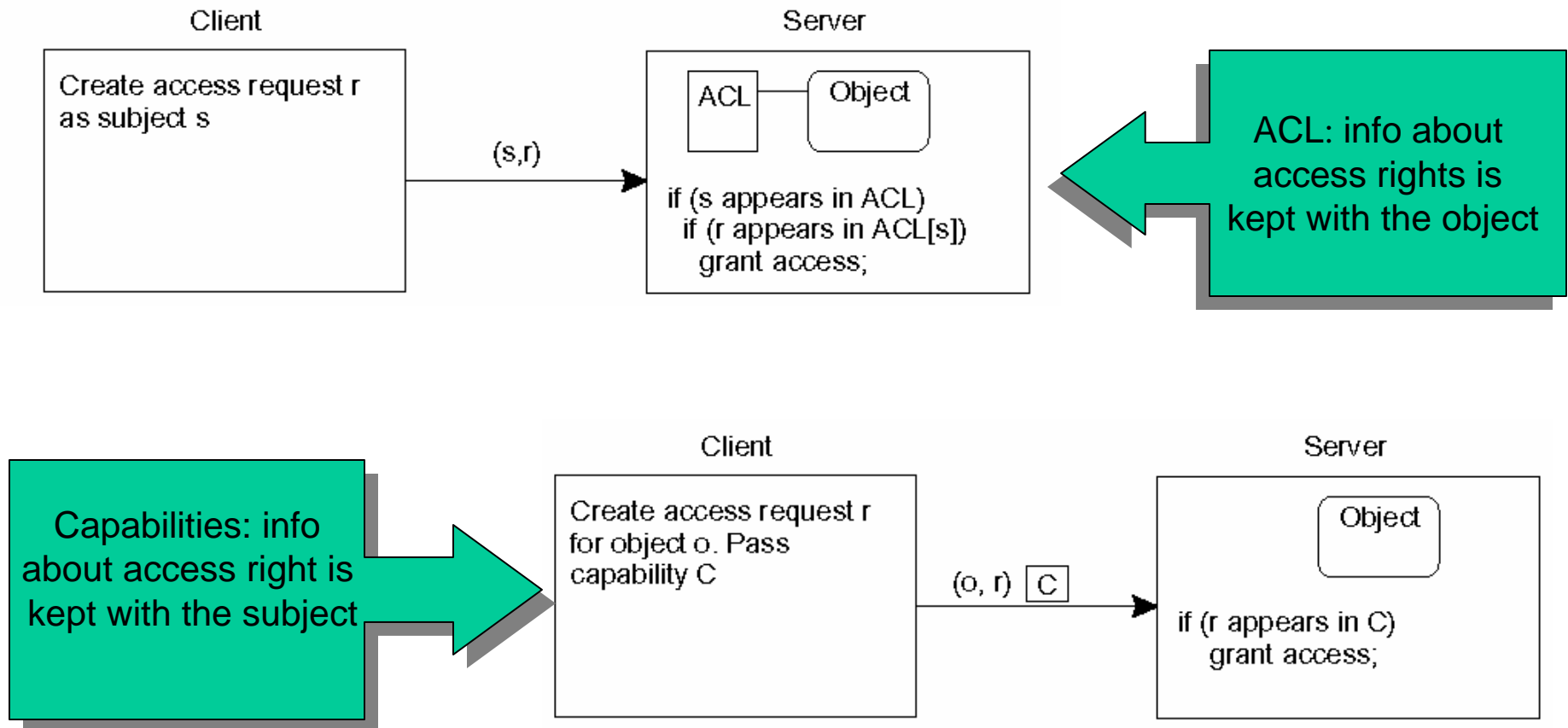
# ACL protection: pros and cons

- Pros:
  - easy revocation: simply remove subject's entry
  - easy review: immediate object→ subjects connection
- Cons:
  - difficult to determine what objects a subject can access
  - list must be accessed at every access (efficiency issue)
  - list can grow very large (storage issue)

# Optimizations for ACLs

- The efficiency issue can be solved caching the subject's access rights after the first access
  - *shadow register* is the place used to store the rights
  - shadow register acts like a capability for the subject
  - disadvantage: register needs to be flushed in case of revocation of access rights
- The storage issue is often solved lumping users in *protection groups*
  - ACL refers to groups and their access rights
  - disadvantage: coarser granularity in assigning rights

# ACL vs. capabilities comparison



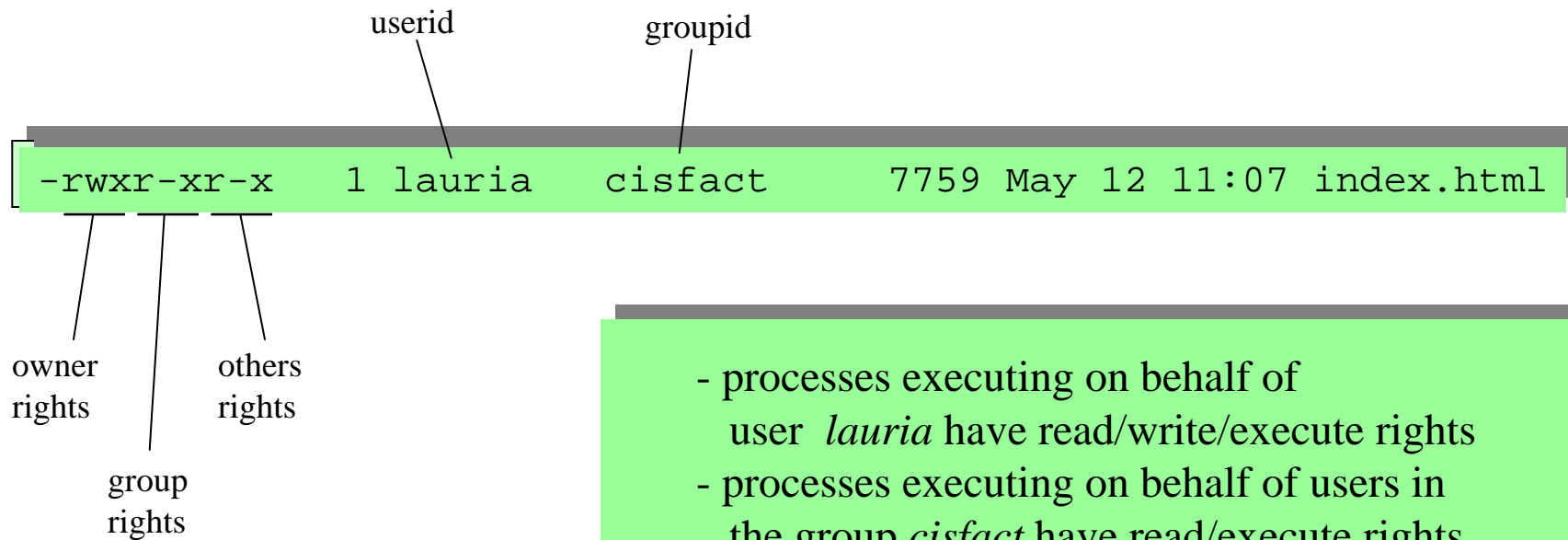
# The Lock-Key method

- This is a combination of previous two methods
  - each subject holds a list of capabilities of the form  $(o, k)$  meaning it can access  $o$  using the key  $k$
  - each object has a ACL that contains lock entries of the form  $(l, \Psi)$  meaning any subject that can open the lock  $l$  can access this object in modes contained in the set  $\Psi$
  - Easy revocation - delete lock entry
  - ACL must still be searched for every access
- Real world example: a similar scheme has been used in the IBM/360 (called the *storage keys* protection method)

# Case studies: Unix

- Protection mechanism: ACL with protection groups
  - protection groups: *owner, group, others*
  - access rights: *read, write, execute*
- Every user is identified by a *userid*,
  - it also belongs to a user group identified by a *groupid*
  - processes execute on behalf of a user, so have a *userid/groupid*
- Main protected entities: files and directories
  - a unique owner is defined for every file/dir (its creator, typically)
  - the file/dir stores the *userid* and *groupid* of the owner
- Access rights are determined by comparing the *uid/gid* of the process requesting access with those of the file

# Unix example

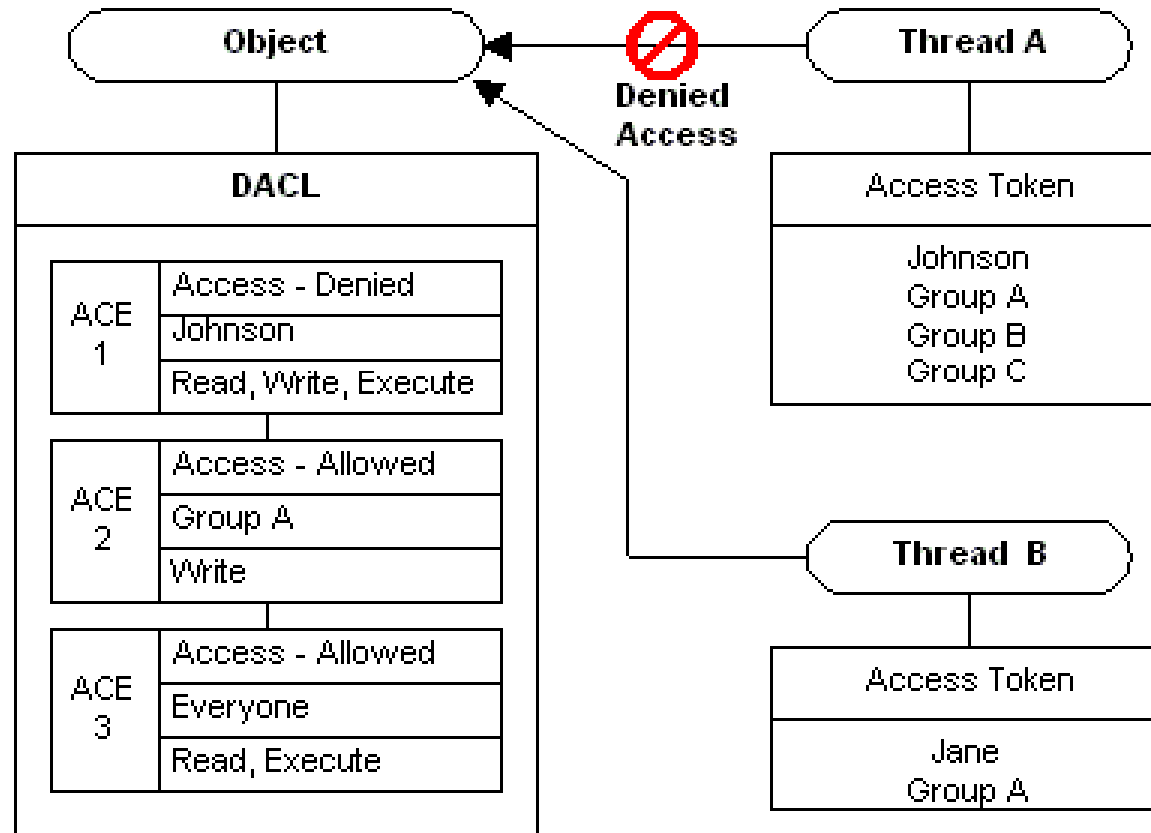


- processes executing on behalf of user *lauria* have read/write/execute rights
- processes executing on behalf of users in the group *cisfact* have read/execute rights
- all others have read/execute permissions

# Case studies: Windows NT

- Protection mechanism: ACL with protection groups
  - the protection groups are created by the sysadm
  - three levels of access rights: generic, standard, object specific
    - generic: GENERIC\_READ, GENERIC\_WRITE, GENERIC\_EXECUTE
- each user is identified by a *security ID (sid)*, can belong to any number of protection groups
  - processes execute on behalf of a user, so have a sid/groupid's (kept in a structure called *access token*)
- Main protected entities: dir/files (others: processes, threads, synchronization objects, network shares, remote printers, ...)
  - each protected object has a full ACL
  - *security descriptor* contains ACL, owner's sid and groupid's

# NT example



- DACL: Discretionary Access Control List; ACE: Access-Control Entries
- Access Token: user and group SIDs

# Case studies: Amoeba

- Amoeba: research distributed operating system developed at Vrije Universiteit, Amsterdam
- Object-based system, client-server architecture
  - client processes execute operations on objects by sending requests to server processes via remote procedure calls
  - capabilities used to control access to objects
  - encryption used to protect capabilities from forgery
  - capability scheme:

48 bits	24 bits	8 bits	48 bits
Server port	Object number	Rights	Check