Windows 2000 Security

Objectives

This is not a Windows security crash course.

Windows security discussed to show how general security principles work in practice.

Understanding the principles will help you to master practical details, should you need to.

Details of Windows security keep changing as the product develops; principles are more stable.

Many features exits to help administration of large systems; this lecture does not teach administration.

Agenda

Windows architecture

Principals & Domains

Subjects

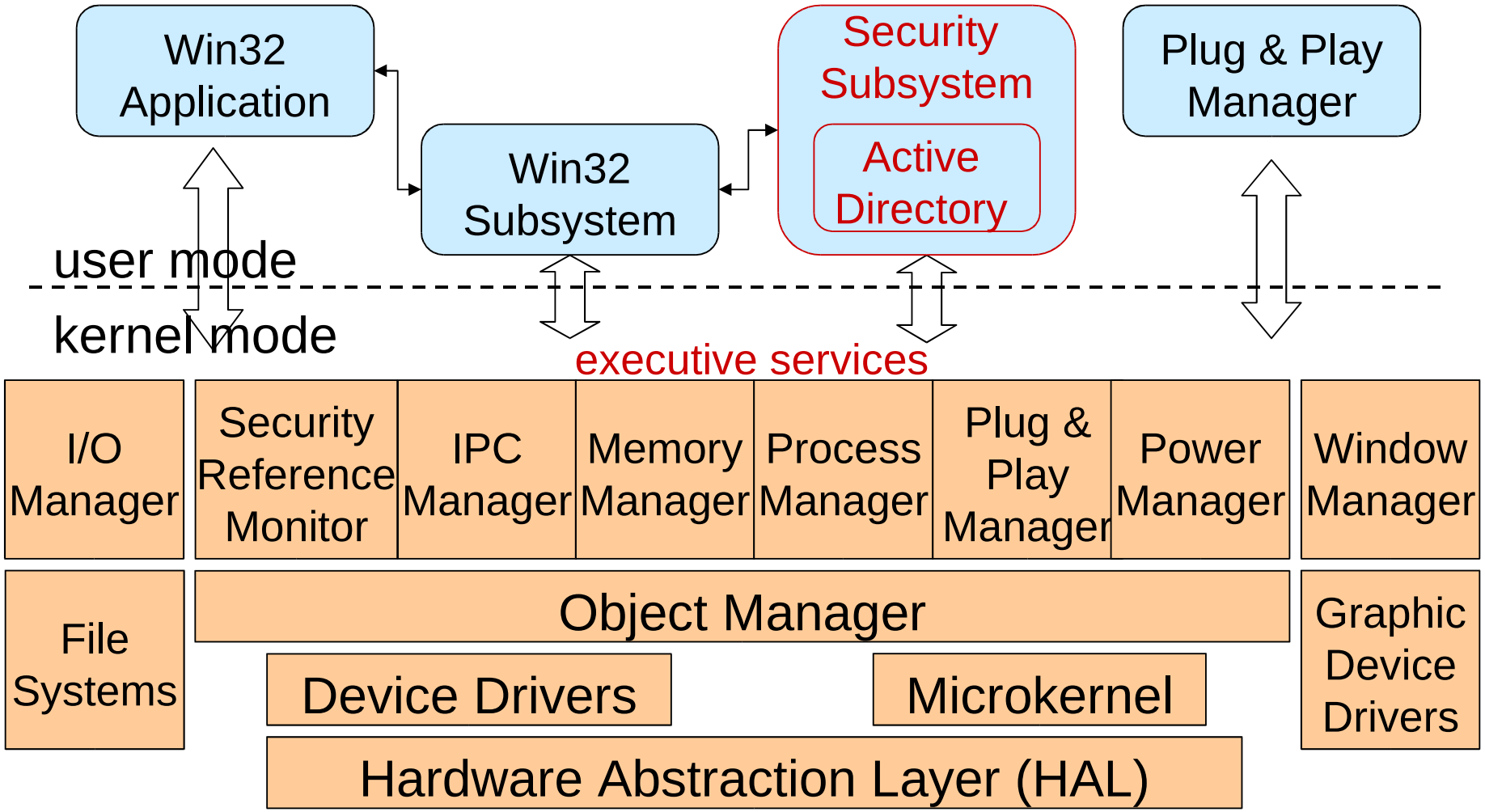
Objects of access control

Privileges & Permissions (access rights)

Access control rules

Managing complexity

Windows Architecture



Windows Architecture

Two modes: user mode & kernel mode

Security components in kernel mode:

- Security Reference Monitor

Security components in user mode:

- Log-on process (WinLogon)
- Local Security Authority (LSA):
  deals with user logon and audit logs
- Security Accounts Manager (SAM): accounts database, including e.g. passwords (encrypted)

Device drivers (often third party products) are running in kernel mode

Registry

Registry: central database for Windows configuration data.

Entries in the registry are called keys.

Registry Editor (Regedit.exe or Regedt32.exe): tool for modifying the registry.

A registry hive is a group of keys, subkeys, and values in the registry.

Registry – Top Level

HKEY_CLASSES_ROOT: contains file extension associations; e.g., you could specify that .doc files are handled by Word.

HKEY_CURRENT_USER: configuration information for the user currently logged on.

HKEY_LOCAL_MACHINE: configuration information about the local computer.

HKEY_USERS: contains all actively loaded user profiles on the system.

HKEY_CURRENT_CONFIG: information about hardware profile used by the local computer at system startup.

Registry

Security relevant hives are

- HKEY_LOCAL_MACHINE\SAM
- HKEY_LOCAL_MACHINE\Security
- HKEY_LOCAL_MACHINE\Software
- HKEY_CURRENT_CONFIG
- HKEY_USERS\DEFAULT

Attack: change registry entry to modify behaviour of the operating system.

Example: keys pointing to locations where the operating system automatically looks for certain executable files; if the permissions set for such a key are weak, malicious software can be inserted by modifying the location.

Registry

Defence: remove the Registry Editor from all machines not used for system management.

Defence: Change security relevant keys through specific utilities

Undefined keys are a potential pitfall.

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecurePipeServers\Winreg specifies who can access the register remotely.

If the key exists, it will be consulted when a user requests remote access to the registry; if the key does not exist, no checks specific to remote access will be performed.

Windows Domains

Stand-alone Windows computers usually administer locally by users; impossible in large organizations.

Domains to facilitate single-sign on and centralized security administration.

Domains can form a hierarchy.

A server can act as domain controller (DC); other computers join the domain.

Domain admins create and manage domain users and groups on the DC.

Active Directory

Directory service in Windows 2000

Hierarchy of typed objects.

Each object type has specific properties.

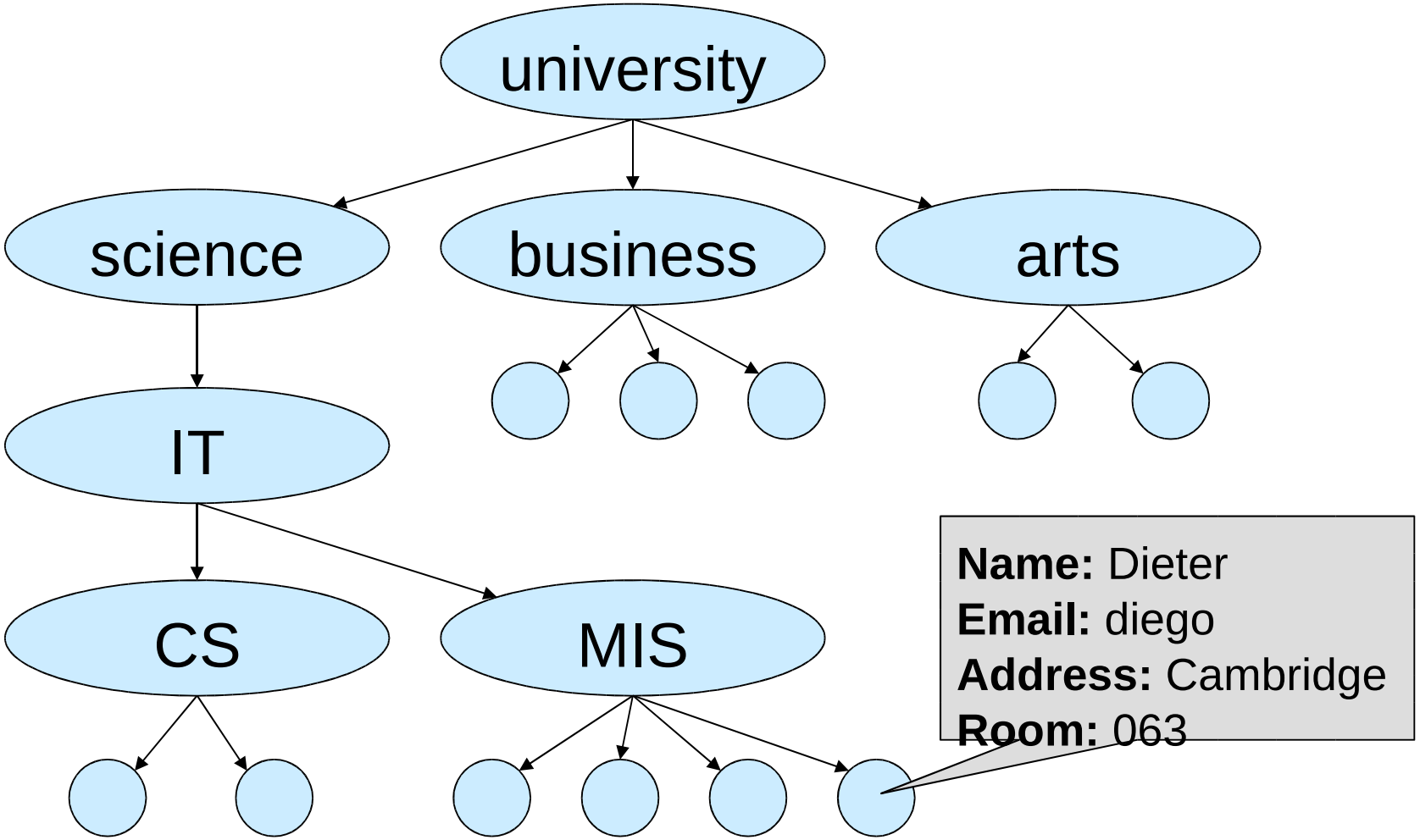
Each object type has a unique GUID (globally unique identifier).

Each property has its own GUID.

Containers: objects that may contain other objects.

Active Directory can be dynamically extended by adding new object types or new properties to existing object types.

Active Directory



Access Control

Access control in Windows applies to objects: files, registry keys, Active Directory objects,…

More complex than access control in a file system.

Means for structuring policies in complex systems: groups, roles, inheritance; access rights beyond read and write.

Identify principals, subjects, and objects.

Access rules: where to find them, how they are evaluated.

Principals

Active entities in a security policy: entities that can be granted or denied access

Principals can be local users, aliases, domain users, groups, machines.

Principals have a human readable name (user name) and a machine readable identifier (SID – security identifier)

Local Security Authority (LSA): each Windows machine has its own built-in authority; users created by the LSA are local users.

Domains & Principals

Domain controller (DC) provides security services for a domain.

The domain controller authority has information about the principal's password and can act as a trusted third party when the principal authenticates itself to some other entity.

Design principle: centralized authentication (password management).

Domains

A domain can have more than one domain controller; updates may be performed at any DC and are propagated using the ‘multimaster replication model’.

Design principle: services decentralized for better performance.

Scoping of Principals

Local principals, administered locally, visible only to the local computer:

- e.g. local system (i.e. O/S), local users

Domain principals, administered by domain admins on a domain controller, seen by all computers on the domain:

- e.g. domain users, Domain Admins alias

Universal principals: e.g. Everyone alias

Where do Principals Live?

Information about principals is stored in accounts and user profiles.

Local accounts are in the Registry (under HKEY_USERS).

Domain accounts are at the Domain Controller but cached locally.

User profile is stored in the file system under "\Documents and Settings\".

Some well-known principals need not be stored anywhere.

Security Identifiers Don't memorize

SID format: S-R-I-SA-SA-SA-N

- S: letter S
- R: revision number (currently 1)
- I: identifier authority (48-bit)
- SA: subauthority (32-bit)
- N: relative identifier, unique in the authority's name space

E.g. Guest S-1-5-21-<*authority*>-501

<*authority*>: 96-bit unique machine or domain identifier created when Windows or domain controller is installed.

Principals – Examples

World (Everyone) S-1-1-0

SYSTEM S-1-5-18

the O/S on a machine runs locally as S-1-5-18; to other machines in the domain the machine is known under a separate, domain specific, SID

Administrator S-1-5-21-<*local authority*>-500

user account created during O/S installation

Administrators S-1-5-32-544

built-in group with administrator privileges, contains initially only the Administrator account

Domain Admins S-1-5-21-<*domain authority*>-512

global group, member of the Administrators alias on all machines in the domain

Principal Names

Principals and authorities have also user-readable names.

Domain users, groups, aliases, machines:
principal@domain = DOMAIN\principal

Local users and aliases:
principal = MACHINE\principal

Example:
diego@europe.microsoft.com = EUROPE\diego
MSRC-688432\Administrators

Principals for Access Control

SID: an individual principal

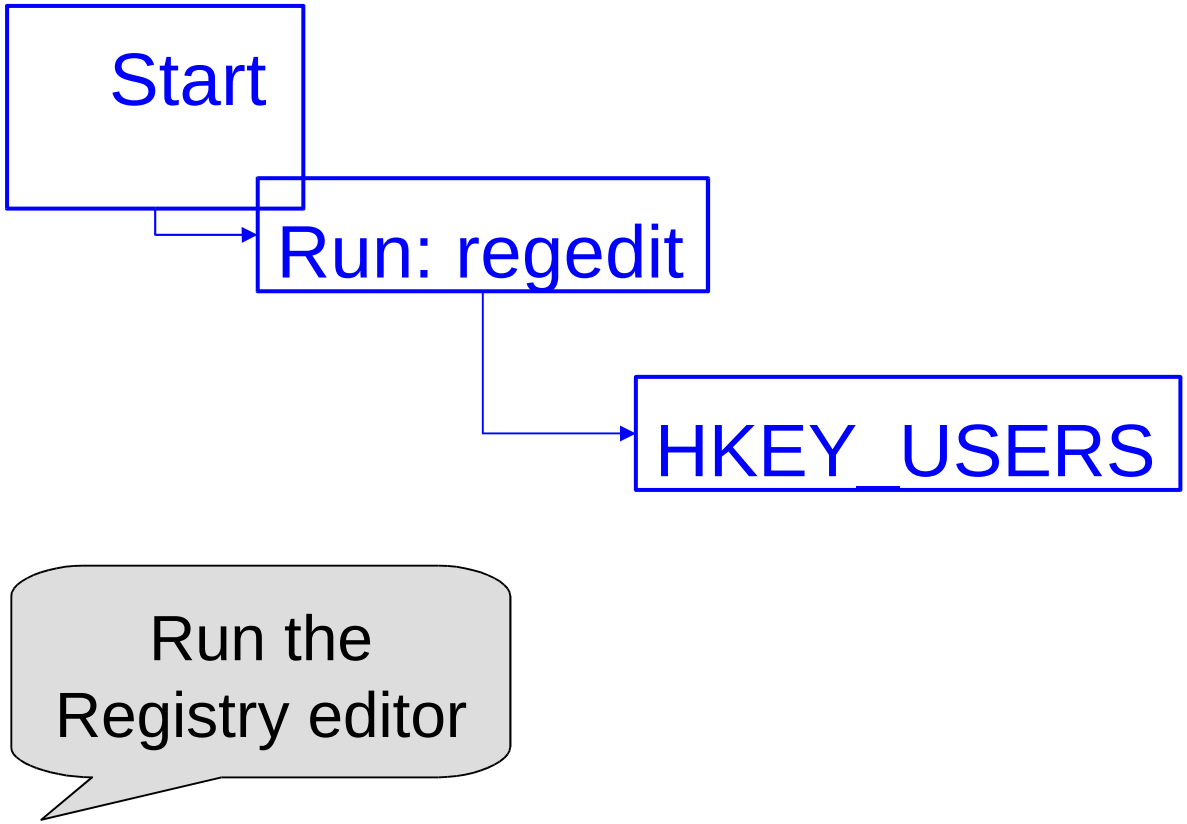
Group: a collection of SIDs managed by the domain controller; a group has its own group SID, so groups can be nested

Alias (local group): a collection of user and group SIDs managed by DC or locally by the LSA;
cannot be nested

Aliases used to implement logical roles: application developer refers to an alias Student, at deployment time appropriate SIDs are assigned to this alias

Aliases implement roles but are not ‘roles’ as defined earlier in the section on RBAC.

Display SIDs on a Machine



Display Principals

Local users and aliases:

> *net user*

> *net localgroup*

Domain users, groups and aliases:

> *net user /domain* **slow!**

> *net group /domain*

> *net localgroup /domain*

Members of a group, e.g.:

> *net group "UK Employees" /domain*

User information:

> *net user diego /domain*

Subjects & Tokens

Subjects are the active entities in the operational system.

In Windows, processes and threads are subjects.

Security credentials for a process (or thread) are stored in a token.

Token contains a list of principals and other security attributes.

New process gets a copy of the parent's token; can restrict it.

Token Contents

Identity and authorization attributes:

- user SID, group SIDs, alias SIDs
- privileges

Defaults for new objects:

- owner SID, group SID, DACL

Miscellaneous:

- logon session ID,…

Some of these fields are read-only, others may be modified

Privileges

Privileges control access to system resources.

Uniquely identified by programmatic name (`SeTcbPrivilege`), have display name ("Act as part of the operating system"), cached in tokens as a locally unique identifier (LUID).

Assigned on a per machine basis.

Assigned to users, groups and aliases.

Different from access rights, which control access to 'securable objects' (explained later).

Privileges – a Few Examples

Back up files and directories

Generate security audits

Manage and audit security log

Take ownership of files and other objects

Bypass traverse checking

Enable computer and user accounts to be trusted for delegation

Shut down the system

Performance and Reliability

Group and alias SIDs are cached in the token, as is the union of all privileges assigned to these SIDs.

Token will not change even if a membership or privilege is revoked.

Better performance.

Better reliability because process can decide in advance whether it has sufficient access rights for a task.

Creating Subjects

The machine is always running a logon process (*winlogon.exe*) under the principal SYSTEM.

When a user logs on to a machine,

- the logon process collects credentials (e.g. user password) and presents them to the LSA,
- The LSA (*lsass.exe*) verifies the credentials,
- the logon process starts a shell (*explorer.exe*) in a new logon session under the user (principal).

Shell spawns processes to the same logon session.

Logging off destroys the logon session and all processes in it.

Authentication in Practice

Authentication binds a subject to a principal.

Most system use passwords for authentication.

Machines are principals and have passwords.

Password authentication can be replaced by other mechanisms, e.g. smart cards.

Pressing CTRL+ALT+DEL provides a trusted path from the keyboard to the logon process.

Creating more Subjects

A process can spawn a new local process (subject) by calling `CreateProcess`.

Each process has its own token: different processes within a logon session can have different credentials

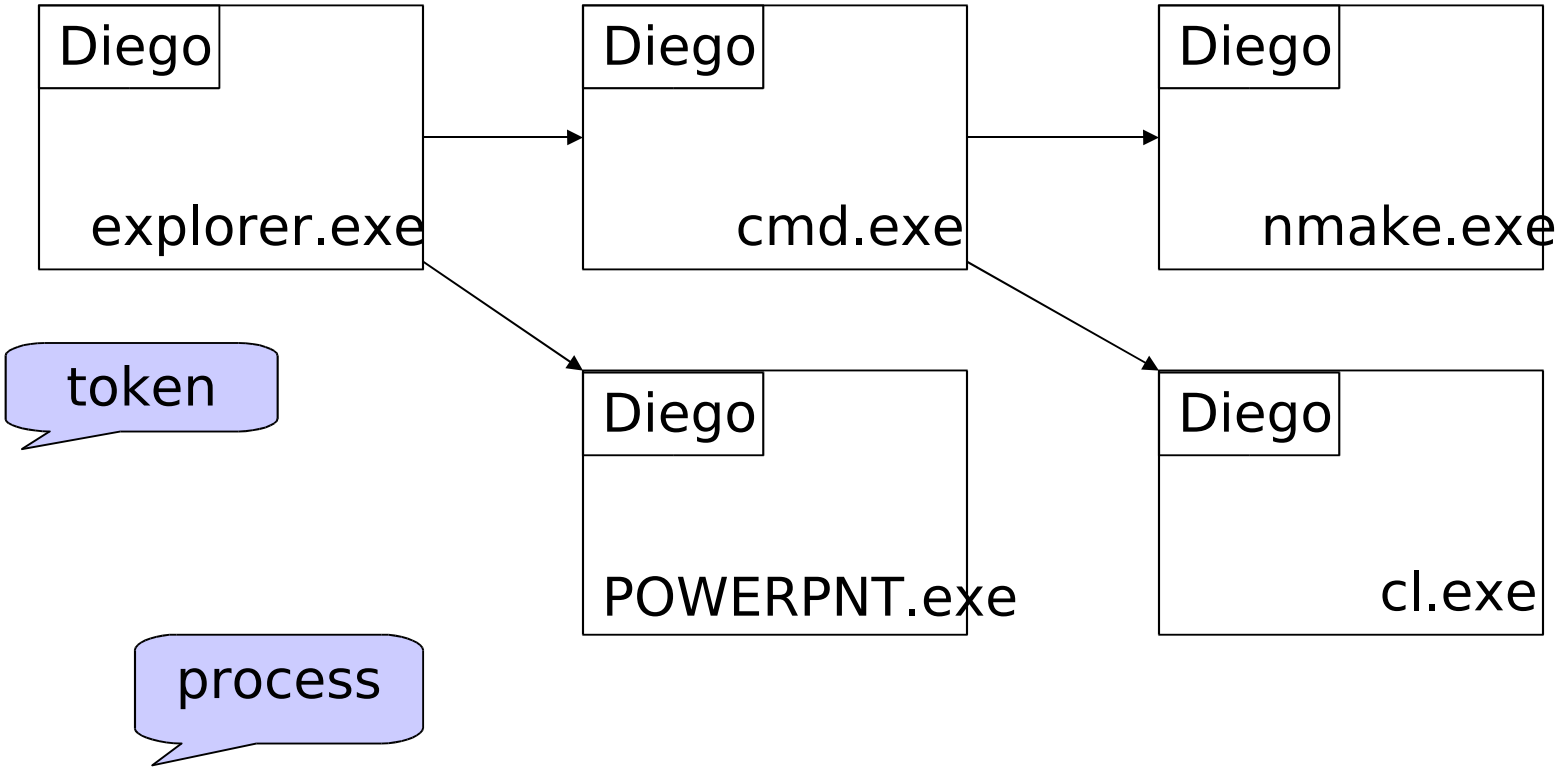
New process gets a copy of parent's token.

Threads can be given different tokens.

User's network credentials (e.g. password) are cached in the interactive logon session.

Processes can create network logon sessions for that user at other machines; network logon sessions do not normally cache credentials.

Processes in a Logon Session



Objects

Executive objects: processes, threads,…

File system objects: files, directories.

Registry keys, printers, …

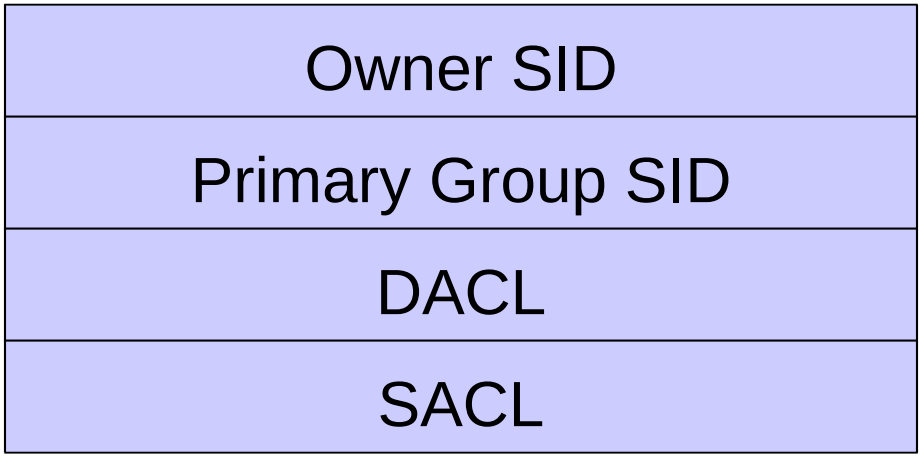
Securable objects have a security descriptor.

Security descriptors for built-in objects are managed by the O/S.

Security descriptors for private objects have to be managed by the application software.

Creating securable objects it is tedious but enables highly granular access control.

Security Descriptor (SD)



Owner: (discussed below)

Primary Group: for POSIX compatibility

DACL: lists who is granted or denied access

SACL: defines audit policy

Permissions (access rights)

Describe what one can do with an object.

Permissions are encoded as 32-bit masks.

Standard permissions

- DELETE
- READ_CONTROL: read access (to security descriptor) for owner, group, DACL
- WRITE_DAC: write access to DACL
- WRITE_OWNER: write access to owner
- SYNCHRONIZE

Specific permissions can be tailored to each class of objects.

Generic Permissions

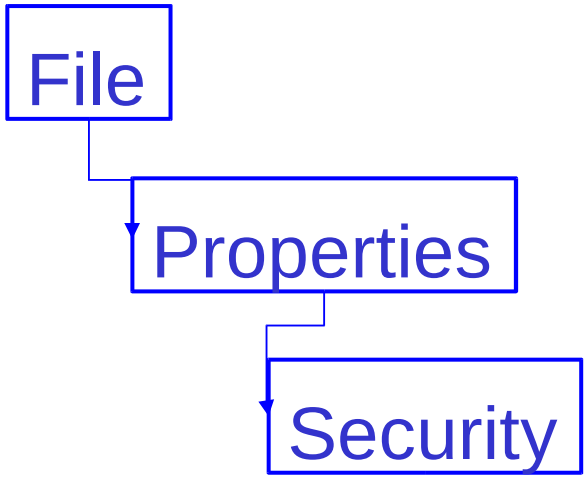
Generic permissions:

- GENERIC_READ
- GENERIC_WRITE
- GENERIC_EXECUTE
- GENERIC_ALL

Each class of objects has a mapping from the generic permissions onto real permissions;

Design principle: intermediate description level; no need to remember class specific permissions.

Find Access Rights for a File



Group or user name

SYSTEM

Dieter

Add Delete

Permissions for …	Allow	Deny
Full Control	✓	
Modify	✓	
Read & Execute	✓	
Read	✓	
Write	✓	
Special Permission		

Advanced

The Owner

A security descriptor indicates the owner of the object.

The owner is a principal.

Objects get an owner when they are created

The owner always has READ_CONTROL and WRITE_DAC permission.

Ownership can also be obtained via the privilege ‘Take ownership of files and other objects’ (`SeTakeOwnershipPrivilege`).

Access Control

Access control decisions consider the

- subject requesting access: credentials of the subject, including its principal, are stored in its token,
- object access is requested for: security attributes stored in its security descriptor,
- desired access (access operation): given as an access mask.

Not all three parameters need be considered.

Implemented by the `AccessCheck` API.

Performance and Reliability

Desired access is compared against the subject's token and the object's security descriptor when creating a handle to the object – not at access time.

E.g. changing file DACL does not affect open file handles.

Better performance.

Better reliability because all access control checks are made in advance, before process starts a task (compare with stack walking).

Access Control Models

Several ways to do Windows access control (with increasing granularity and complexity).

Impersonation: access control based on the principal requesting access; process 'impersonates' the user SID of its token; coarse but simple to implement.

Impersonation is a typical O/S concept and does not work well at the application level.

Role-centric: use groups and aliases to give a process suitable access rights for its task.

Object-centric: objects at the application level get a security descriptor; can get complex.

Access Control Lists

DACL in security descriptor: list of access control entries (ACEs).

ACE format:

- Type: positive (grant) or negative (deny)
- Flags
- ObjectType (new in Windows 2000)
- InheritedObjectType (new in Windows 2000)
- Access rights
- Principal SID: principal the ACE applies to

Example 1

ACE1	
Type:	ACCESS_ALLOWED_OBJECT_ACE
Access rights:	write
Principal SID:	PRINCIPAL_SELF
InheritedObectType:	{GUID for User Account objects}
ObjectType	{GUID for web homepage}

Grants users permission to set their own homepage.

The PRINCIPAL_SELF SID represents the user; the application supplies the user's SID when making an access request.

Example 2

ACE2	
Type:	ACCESS_ALLOWED_OBJECT_ACE
Access rights:	create child
Principal SID:	Server Applications
InheritedObectType:	{GUID for RPC Services}
ObjectType	{GUID for RPC Endpoint}

Allows Server Applications to create RPC endpoints in any container of type RPC Services.

ACE will be inherited into any container of type RPC Services.

Access Check

Accumulate permissions: take permissions from owner; go through DACL and check ACEs where the subject's token contains a matching SID:

- grant access once all permissions needed for the requested access are obtained;
- deny access once a relevant negative ACE is found or the end of the DACL is reached.

Typical access rights in an ACE: read, write, create, delete, control

Null DACL

For negative ACEs to take precedence over positive ACEs, they must be placed at the top of the DACL.

For finer granularity of access control, you may want to place negative ACEs after positive ACEs.

Empty DACL: nobody is granted any permission.

No DACL (NULL DACL): everybody gets all access.

Type & ObjectType

Type: says whether access is granted or denied and whether the ACE contains an ObjectType:

- ACCESS_ALLOWED_ACE
- ACCESS_DENIED_ACE
- ACCESS_ALLOWED_OBJECT_ACE
- ACCESS_DENIED_OBJECT_ACE

ObjectType: a GUID defining an object type.

Applications can now include the ObjectType of objects in their access requests.

ObjectType

For a given request, only ACEs with a matching ObjectType or without an ObjectType are evaluated.

Control read/write access on object property: put the GUID of the property in ObjectType.

Control create/delete access on objects: put the GUID of the object type in ObjectType.

ACEs without ObjectType are applied to all objects.

Property Sets

Properties can be collected in property sets; a property set is identified by its GUID.

Ease of administration: instead of ACEs for all the properties of an object type, there is one ACE that refers to the property set.

In an access request, a list of properties can be passed to the reference monitor.

A single check against the property set returns the result for each property.

Changes to the properties of an object type do not force us to change the ACL.

Inheritance

We need systematic and automatic procedures for assigning ACEs to objects.

Typically, when a new object is created its ACEs are inherited from the container (directory) the object is being placed in.

In Active Directory a container may contain objects of different types, therefore a selective inheritance strategy is needed.

Inheritance is controlled through inheritance flags and through the InheritedObjectType.

Inheritance

Static inheritance: ACEs are inherited from the container when a new object is created; later changes to the container have no immediate effect.

ACEs are annotated with a flag that indicated whether they are inherited.

For changes to take effect, a propagation algorithm has to be run.

Idempotence: re-applying the propagation algorithm does not change the access rights.

Inheritance Flags

INHERITED_ONLY_ACE: ACE only used for inheritance, not applied to the object itself.

NO_PROPAGATE_INHERIT: inherited only by the next generation but not propagated further.

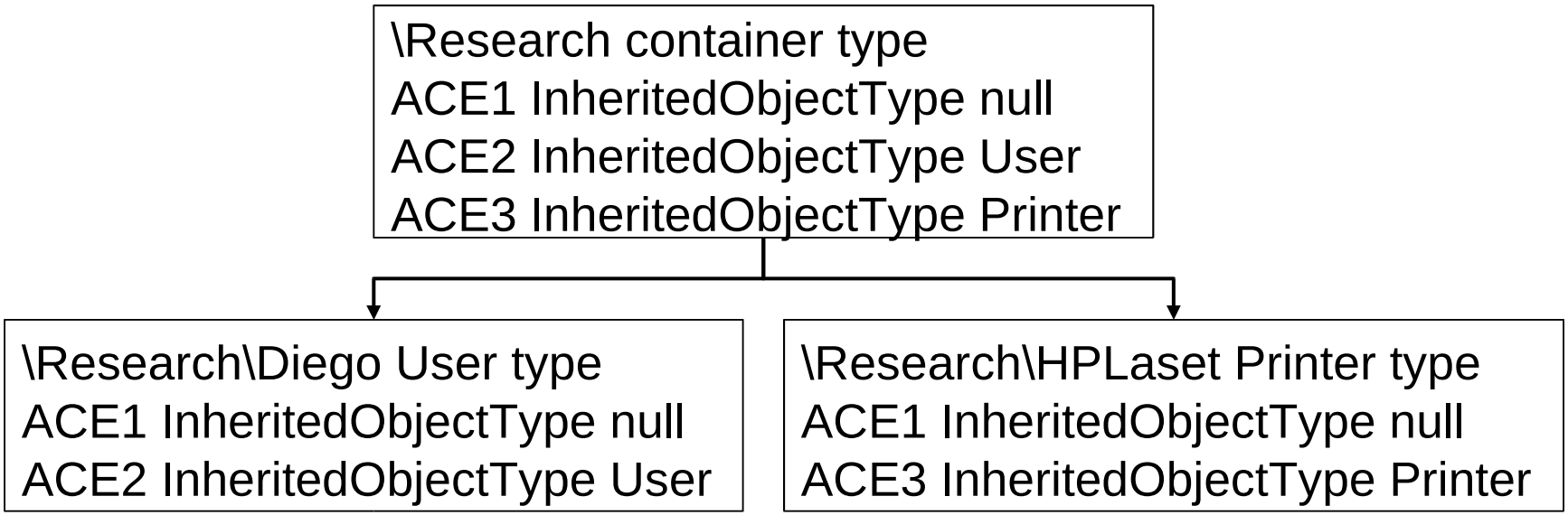
OBJECT_INHERIT_ACE: inherited by all subobjects that are not containers.

CONTAINER_INHERIT_ACE: inherited by subobjects that are containers.

InheritedObjectType

Specifies the object type that inherits the ACE.

When an object is created only ACEs with a matching InheritedObjectType or without an InheritedObjectType are copied into its ACL.



Controlling Inheritance

CREATOR_OWNER: “placeholder” SID, replaced in an inherited ACE by the owner’s SID when an object is created.

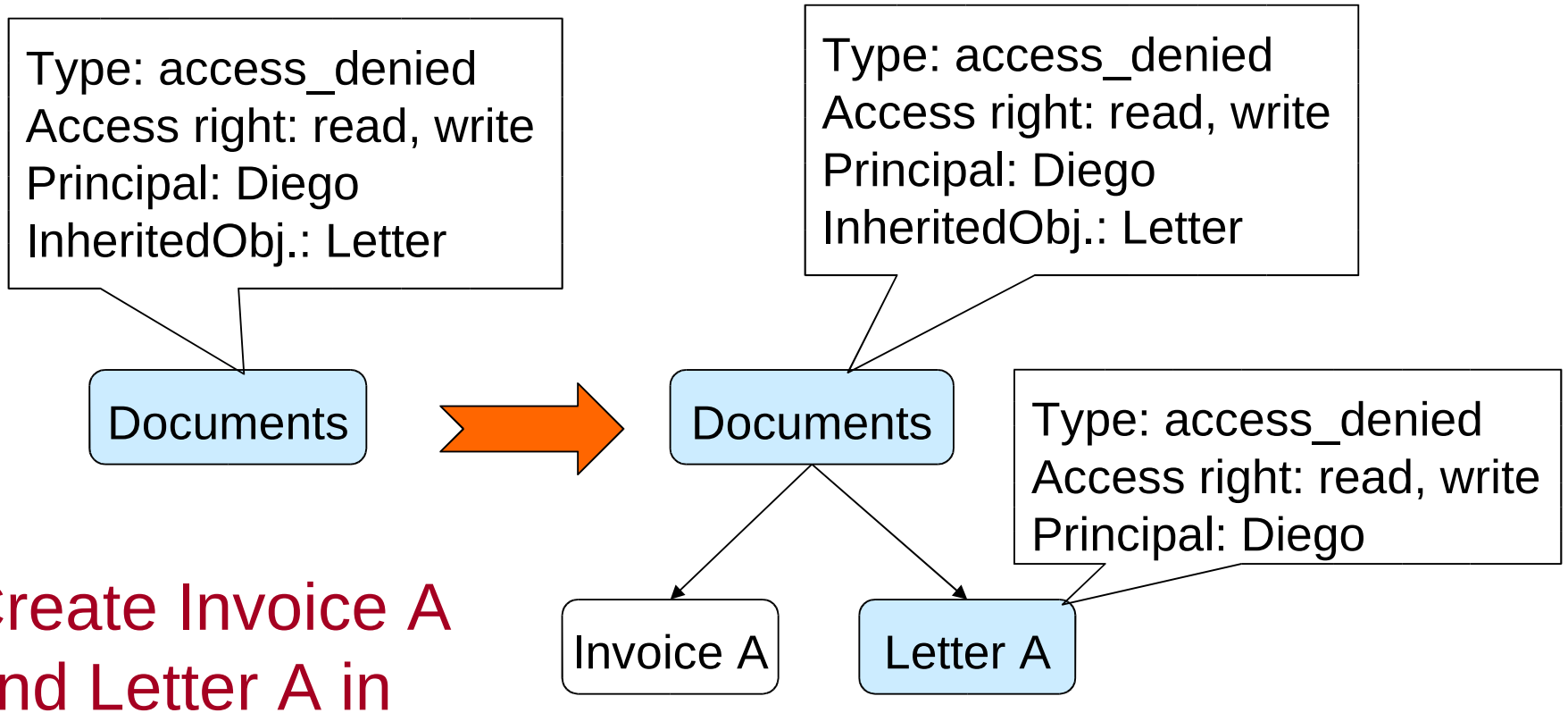
To block ACE inheritance, set the flag SE_DACL_PROTECTED in the security descriptor.

Ordering of ACEs in the DACL: locally added ACEs are placed in front of inherited ACEs.

ACEs from closer containers are placed in front of more distant containers.

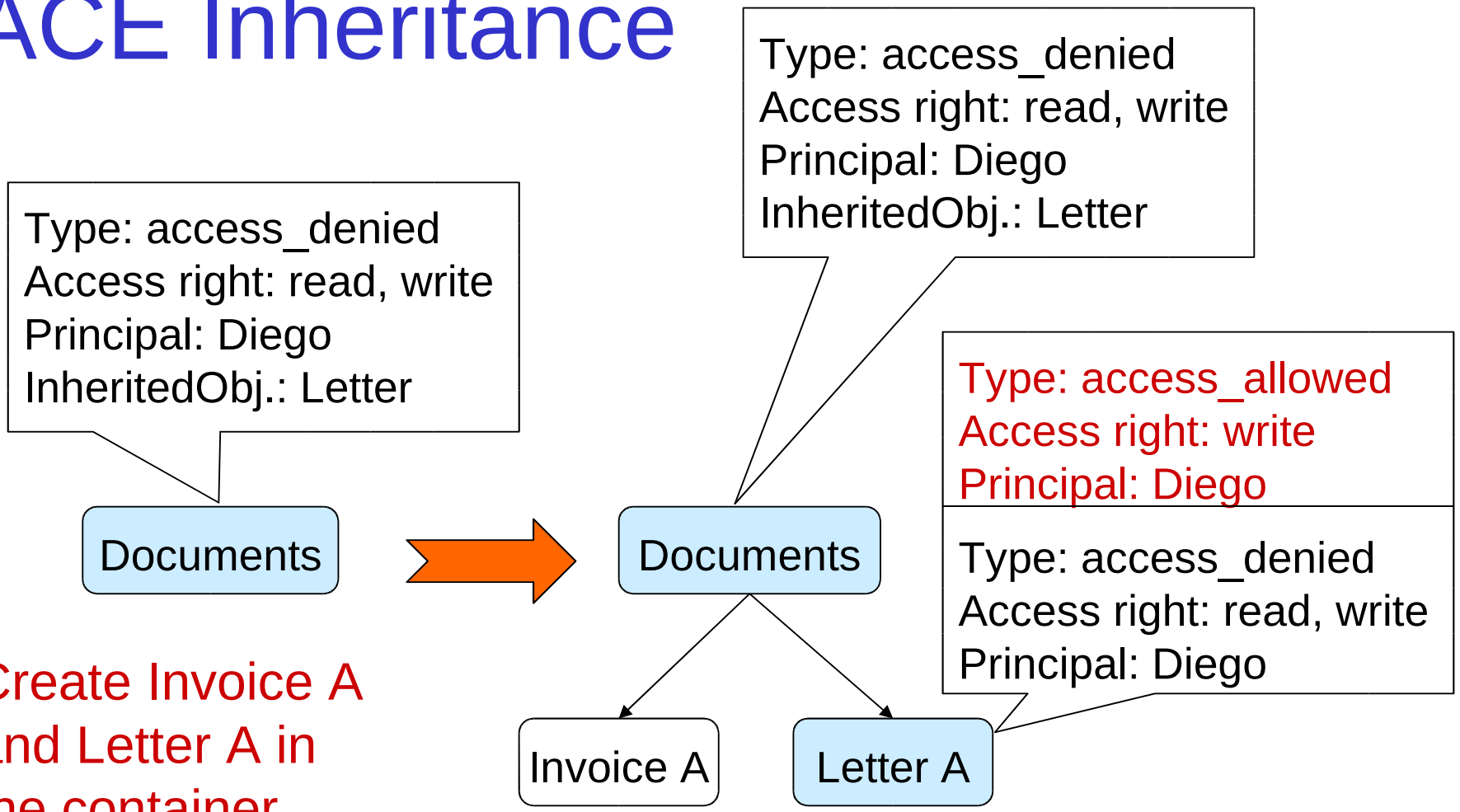
A positive ACE can appear before a matching negative ACE.

ACE Inheritance



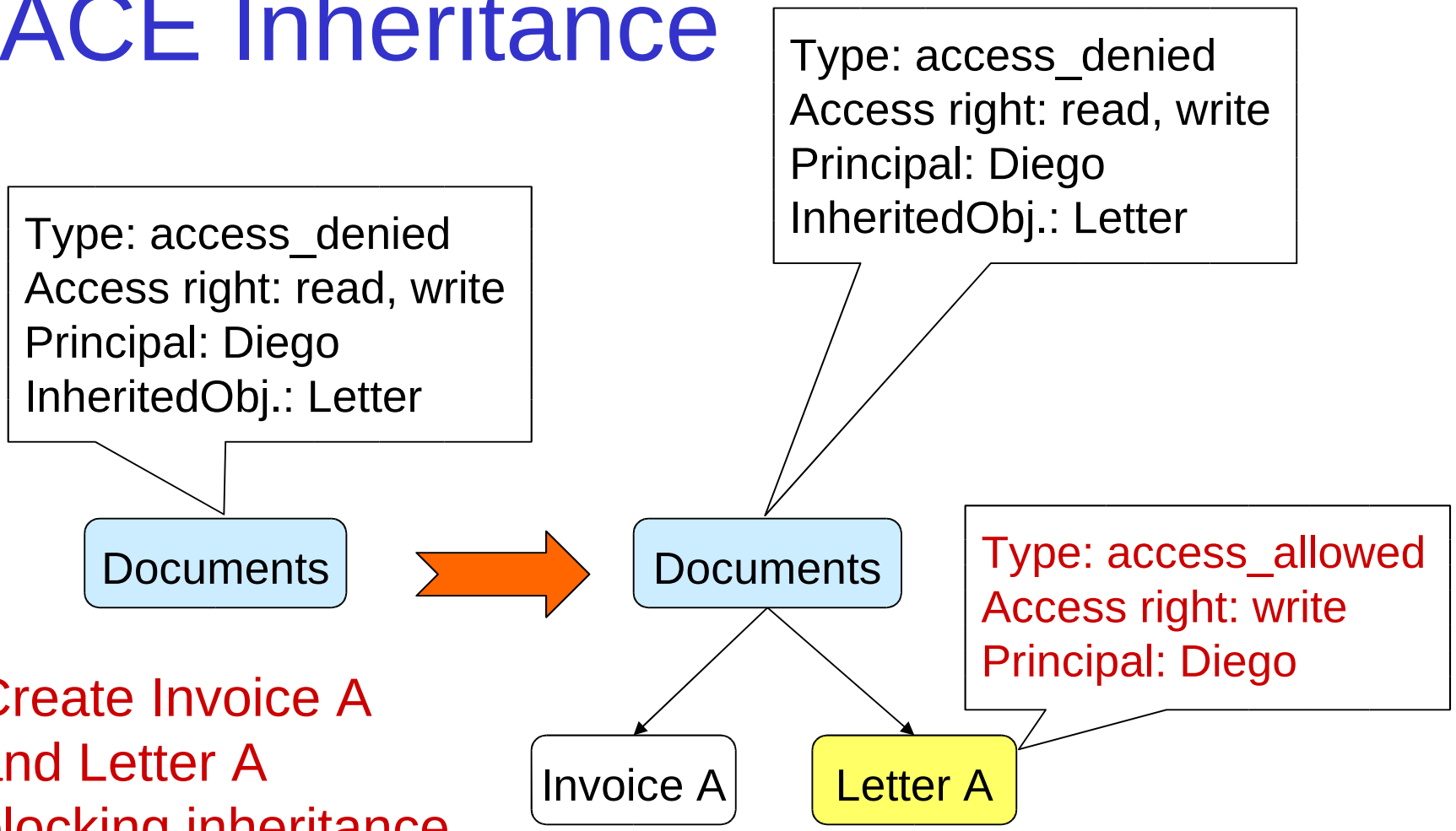
Create Invoice A and Letter A in the container;
Letter A inherits its ACE

ACE Inheritance



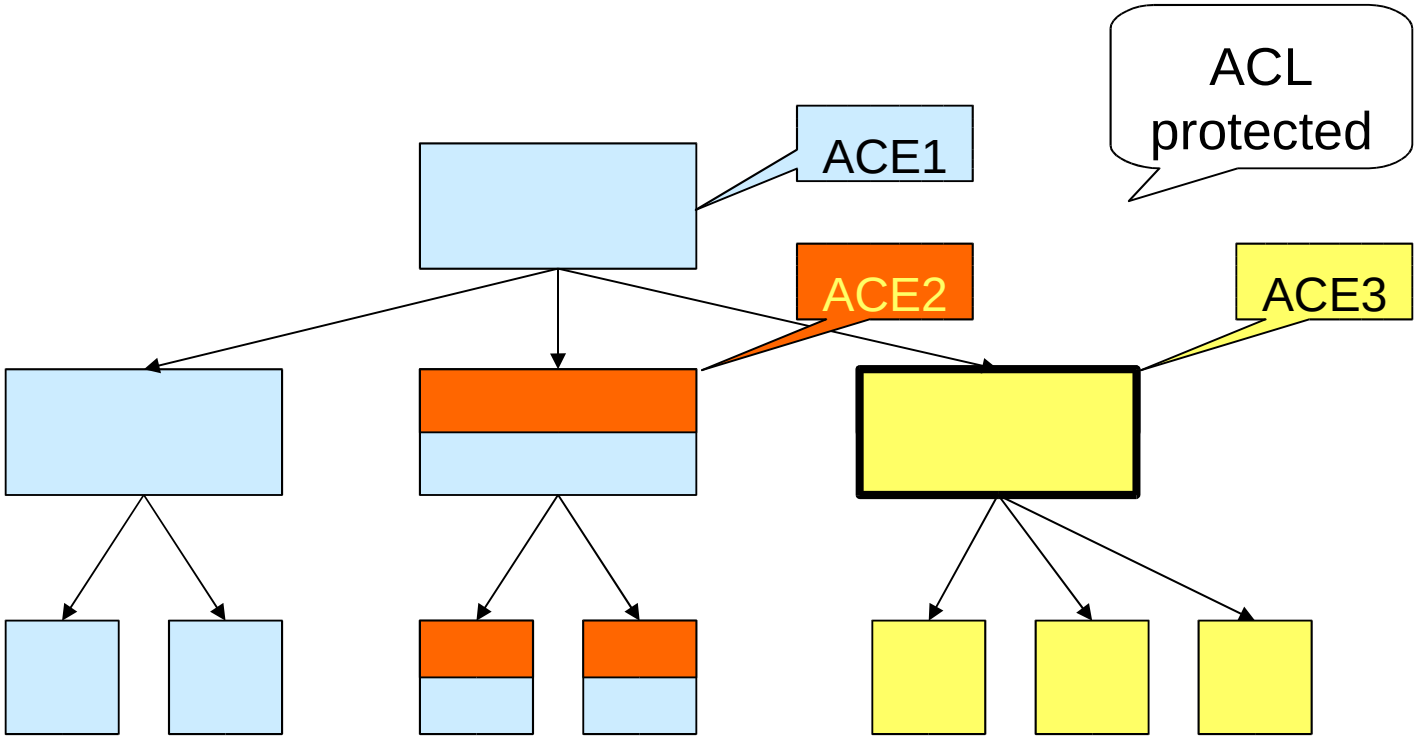
Create Invoice A and Letter A in the container defining new ACE for Letter

ACE Inheritance



Create Invoice A
and Letter A
blocking inheritance
on Letter A with SET_DACL_PROTECTED

ACE Inheritance



ACE1 inherited in the entire subtree

ACE2 inherited in subtree in front of ACE1

ACE3 inherited in the subtree, ACE1 is blocked

Restricted Contexts

So far access control has (implicitly) referred to users.

We may also want to control what certain programs can do.

It is usual but not particularly helpful to refer to such programs as “untrusted code”.

Windows 2000 addresses this issue with restricted tokens.

A process running with a restricted token is a restricted context.

Design principle: least privilege.

Restricted Tokens

Restricted tokens can be constructed from a given access token by

- removing privileges,
- disabling groups: groups are not deleted but marked as USE_FOR_DENY_ONLY,
- adding a restricted SID representing a program; this SID has to be entered into the ACLs of the objects the program should have access to.

Restricted SIDs may be created

- per program and added to the ACLs of required resources (object types);
- per object type and be added to restricted tokens.

Restricted SID

A process with a restricted token gets access only if both the SID and the restricted SID are granted access

User SID	Dieter
Group SIDs	Administrators *use for deny only* Users
Restricted SIDs	MyApp
Privileges	(none)

Ace 1:
Access Rights: read, write
Principal SID: Dieter

Ace 2:
Access Rights: read
Principal SID: MyApp

Ace 1:
Access Rights: read
Principal SID: Admin

Ace 2:
Access Rights: read
Principal SID: MyApp

Ace 1:
Access Rights: read
Principal SID: Dieter

Managing Security

Starting up: built-in accounts and predefined principals with predefined privileges, such as the Administrator accounts and groups.

Editing: various GUI editors to manipulate and check security attributes.

Managing complexity: inheritance of security attributes; group policies (Active Directory) specified for domains (or site, OU), applied to principals in the domain.

Management tools that give a high level view of the ‘security state’.

Default Accounts

Three types of default user and group accounts.

- Predefined accounts: installed with the operating system.
- Built-in accounts: installed with the operating system, application, and services.
- Implicit accounts: created implicitly when accessing network resources.

Default users and groups created by the operating system can be modified, but not deleted.

LocalSystem is a built-in account used for running system processes and handling system-level tasks; users cannot log-in to this account, but certain processes can do so.

Adminstrator & Guest

Administrator and Guest are predefined accounts installed locally.

The Administrator account cannot be removed or disabled; has complete access to files, directories, services, and other facilities.

In a domain, the local Administrator account is primarily used when the system is first installed; once installation is completed, the actual administrators can be made members of the Administrators group.

Guest account: intended for users who only need occasional access; when Windows 2000 is installed, the Guest account is disabled.

Built-in Groups

Have predefined user rights and permissions and provide another level of indirection when assigning access rights to users; users obtain standard access rights by becoming member of such a built-in group.

Typical examples: Administrators, Backup Operators, User, or Guests.

System managers should stick to the built-in groups when implementing their security policies and define groups with different permission patterns only if there are strong reasons for doing so.

Implicit Groups

Everyone: contains all local and remote users, including Guest; this group can be used to grant or deny permissions to all users.

Interactive: contains all users logged on locally.

Network: contains all users logged on over the network.

System: the operating system.

Creator Owner: the creator or owner of a file or a resource.

Managing Security: Audit

Audit rules coded in the SACLs, which are non-discretionary and set by administrators.

ACE format in SACL:

- Type: positive (audit grant of permission) or negative (audit denial of permission)
- Trustee: a SID (individual, group, alias)
- Mask: permission (32-bit mask)

An ACE can be both positive and negative.

Audit events are not generated by the access control function but by special audit functions.

Managing Security: Updates

Remember: security is a moving target.

New vulnerabilities keep being detected.

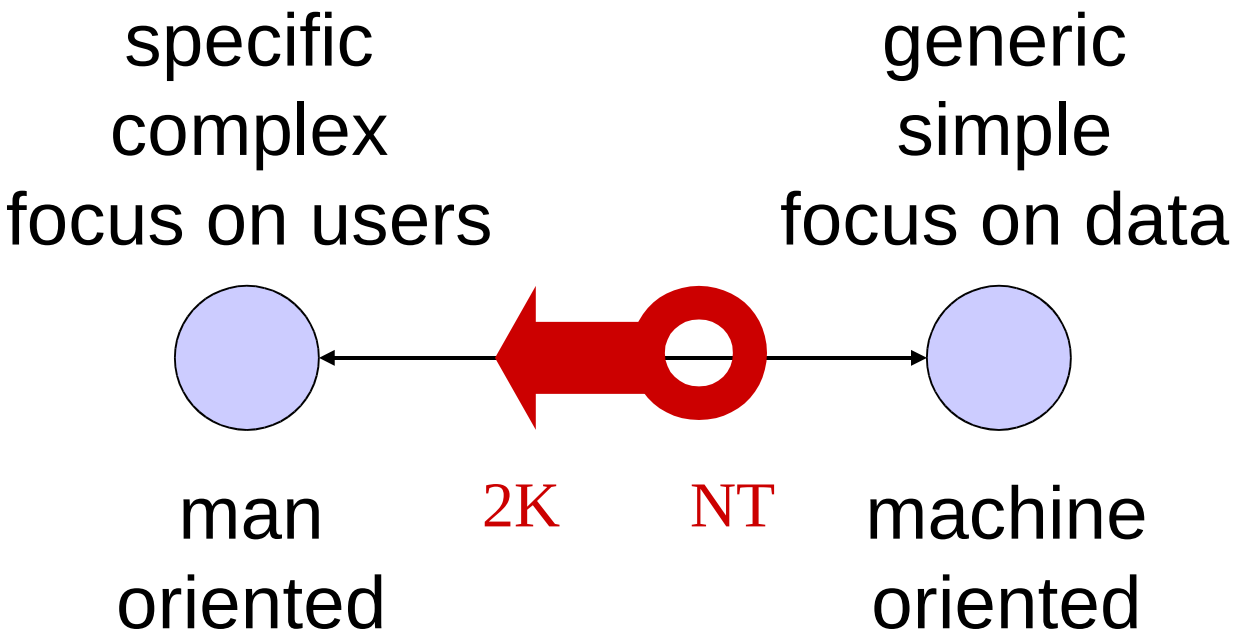
Patches become available either individually or in Service Packs.

Security patches have to be installed: often attacks exploit vulnerabilities where patches have been available for some time.

Systems managers have to keep up-to-date with current threats.

CERT advisories and the like: www.cert.org, www.sans.org, www.securityfocus.com

Man-machine Scale: Windows



Sources Used

Keith Brown: *Programming Windows Security*, Addison-Wesley, 2000 (good general introductions & examples, detailed advice for system programmers; highlights changes from Windows NT)

M. M. Swift et al.: *Improving the Granularity of Access Control in Windows 2000*, ACM Transactions on Information Security, Vol. 5, No. 4, pages 398 – 437, 2002