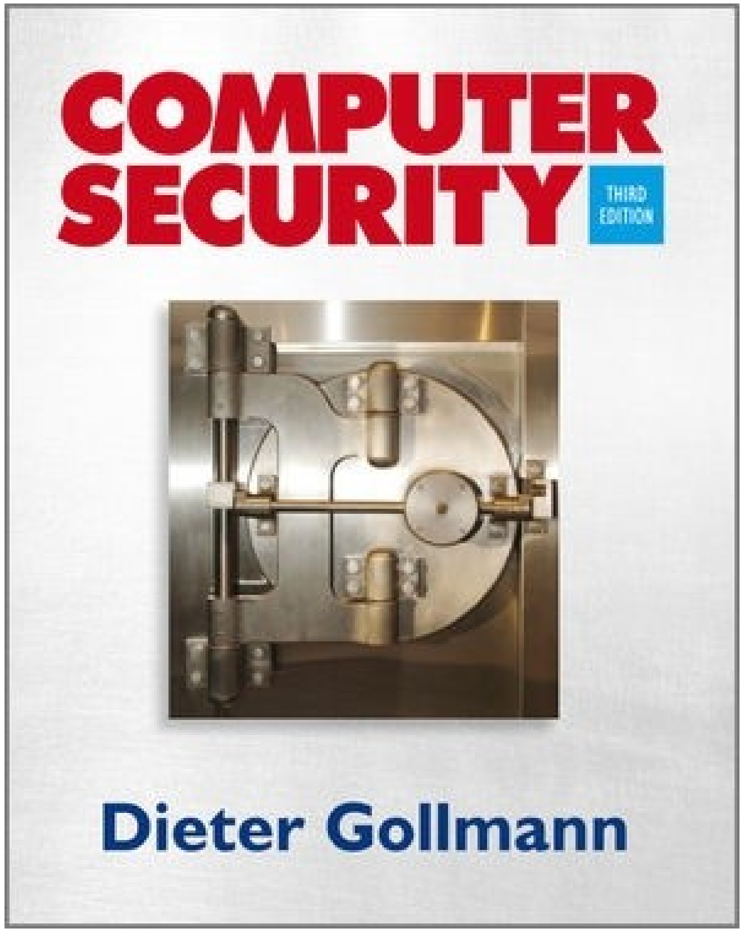
Computer Security 3e

Dieter Gollmann



Chapter 8:
Windows Security

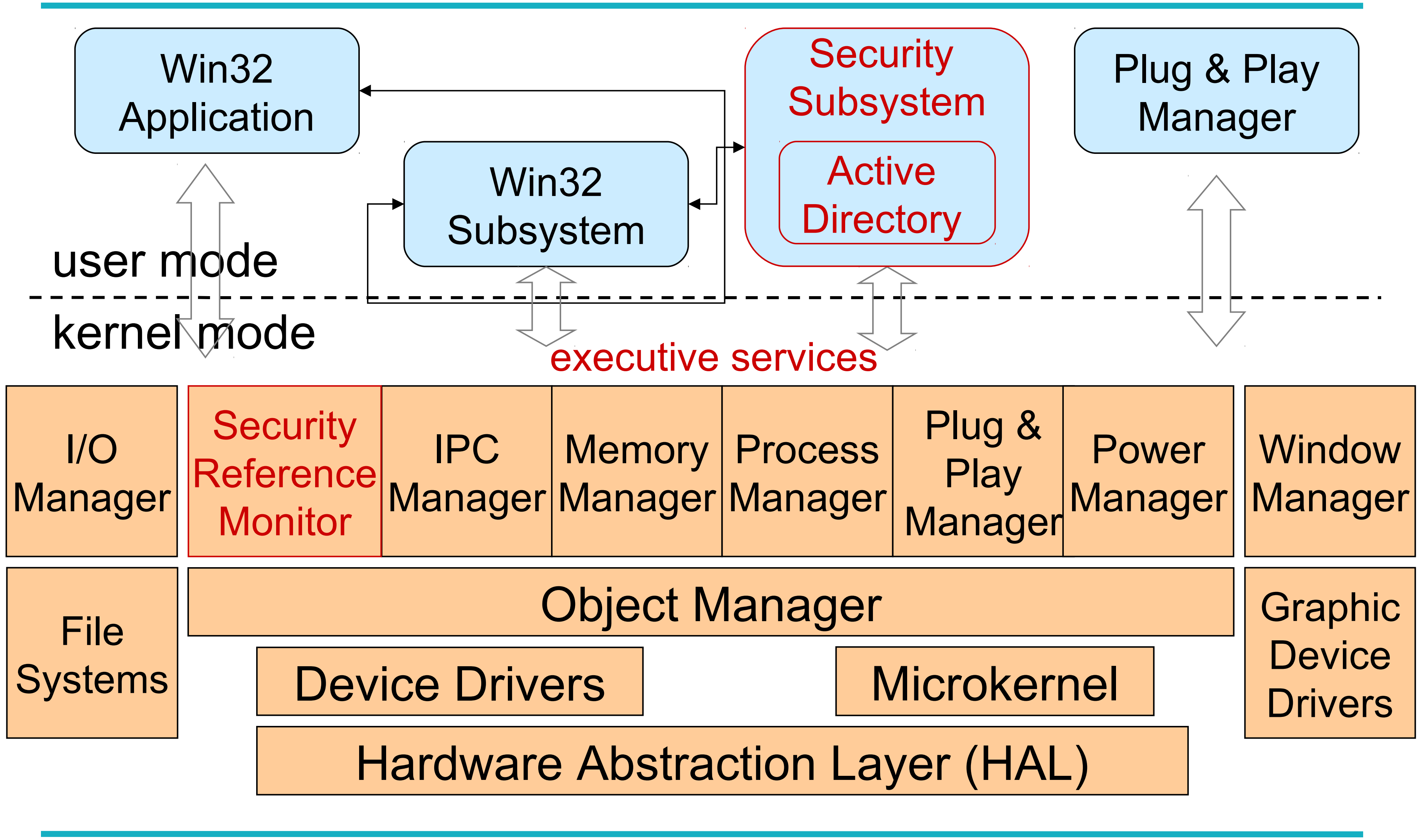
Objectives

- 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
- Least privilege: restricted contexts, UAC

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) run in kernel mode.

Registry

- Registry: central Windows configuration database.
- Entries in the registry are called keys.
- Registry hive: group of keys, subkeys, and values in the registry.
  - HKEY_CLASSES_ROOT: holds file extension associations; e.g., to 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.

Windows Domains

- Stand-alone Windows machines usually administered locally by users; impossible in large organizations.
- Domains facilitate single-sign on and centralized security administration.
- A server can act as domain controller (DC); other computers join the domain.
    - A domain can have more than one DC; updates may be performed at any DC.
- Domain admins create and manage domain users and groups on the DC.
- Domains can form a hierarchy.

Access control

- Access control in Windows applies to objects: files, registry keys (systems database), Active Directory objects,…
- More complex than access control in a file system.
- Access rights beyond read, write, execute.
- Means for structuring policies in complex systems: groups, roles, inheritance.
- Identify principals, subjects, and objects.
- Access rules: where to find them, how they are evaluated.

Principals

- Active entities in a security policy: can be granted or denied access.
- Principals: local users, domain users, groups, aliases, machines.
- Principals have a machine readable security identifier.
- Principals have a human readable user name.
  - Domain users, groups, aliases, machines:
    principal@domain = DOMAIN\principal
    E.g. diego@europe.microsoft.com = EUROPE\diego
  - Local users and aliases:
    principal = MACHINE\principal
    diego@europe.microsoft.com = MSRC-688432\Administrators

Scoping of Principals

- Local Security Authority (LSA): each Windows machine has its own built-in authority; users created by the LSA are local users.
- 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 in domain:
  - e.g. domain users, Domain Admins alias
- Universal principals: e.g. Everyone alias

Security Identifiers

- Security identifier (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
- E.g. World (Everyone) S-1-1-0

SID – Examples

- SYSTEM S-1-5-18
  O/S runs locally as S-1-5-18; in its domain 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 a domain.

Well-known Principals

- Well-known principals have same relative identifier in each domain.
- 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
- Design principle: “Short cut” to placeholder principals.

Morrie Gasser, 1990

Because access control structures identify principals, it is important that principal names be globally unique, human-readable and memorable, easily and reliably associated with known people.

Creating an Authority

- A new issuing authority gets a SID with identifier authority 5, followed by 21 and a 96-bit random number put into three subauthority fields.
- Design principle: authorities have (statistically) unique identifiers.
- SIDs include the identifier of the issuing authority (domain), so a SID cannot by mistake represent access rights in the scope of some other domain.
- Design principle: use randomness for creating unique name spaces.

Creating a SID

- SID constructed when a user account is created, fixed for the lifetime of the account.
- Pseudo-random input (clock value) used to construct a SID; you will not get the same SID if you delete an account and then recreate it with exactly the same parameters as before.
- SIDs for users and groups are unique and cannot be assigned again to another user or group.
- A principal cannot by mistake get permissions of a previous principal.

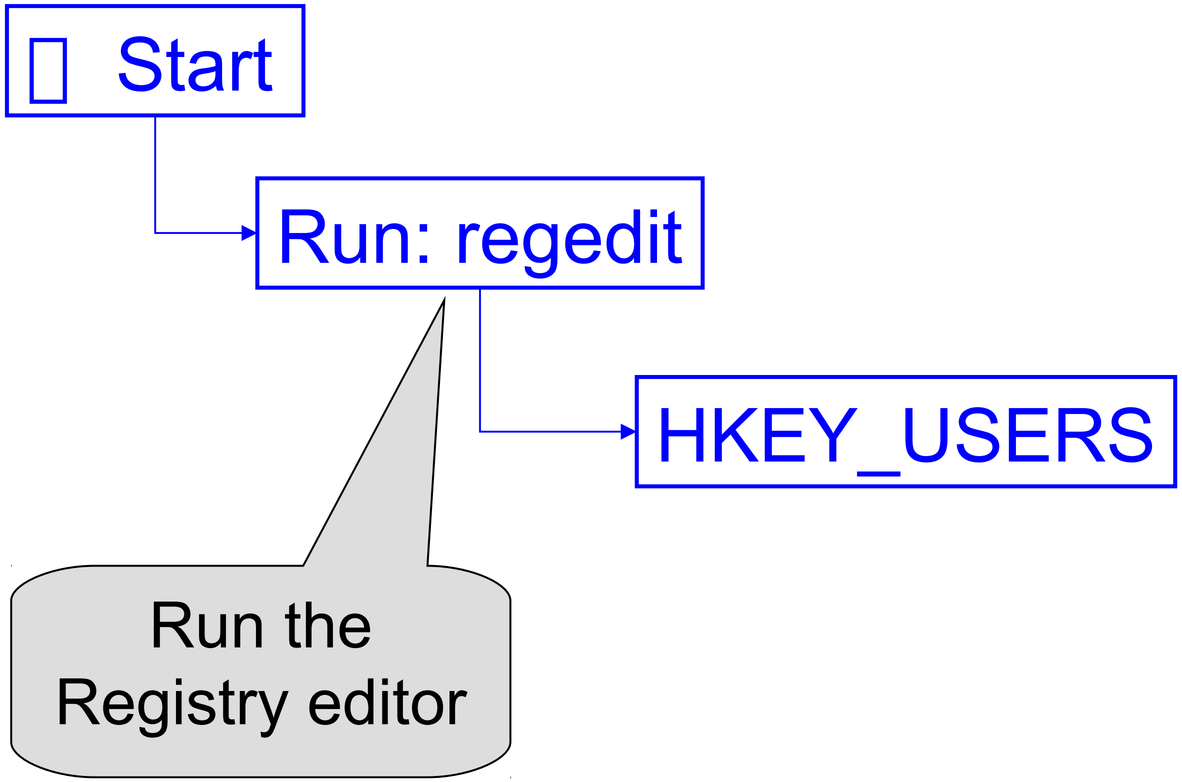
Where do Principals Live?

- Information about principals stored in accounts and user profiles.
- User profiles stored in file system under \Documents and Settings\.
- Local accounts in Registry (under HKEY_USERS).
- Domain accounts at the Domain Controller, also cached locally.
- Domain controller authority knows the principal’s password; can act as a trusted third party when principal authenticates itself to some other entity.
- Design principle: Centralized authentication (password management).

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): collection of user and group SIDs managed by DC or locally by 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.
- Design principle: support instantiation of policies that refer to placeholder principals.

Display SIDs on a Machine



Subjects & Tokens

- Subjects: active entities in the operational system.
- In Windows, processes and threads are subjects.
- Security credentials for a process (or thread) 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 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 to users, groups and aliases.
- Assigned on a per machine basis.
- Different from access rights, which control access to ‘securable objects’ (explained later).

Privileges – Examples

- Back up files and directories
- Generate security audits
- Manage and audit security log
- Take ownership of files and other objects: (`SeTakeOwnershipPrivilege`).
- Bypass traverse checking (Exercise: find out more about this privilege; is it a security problem?)
- 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 as process can decide in advance whether it has sufficient access rights for a task.

Creating Subjects

- A machine is always running a logon process (*winlogon.exe*) under the principal SYSTEM.
- When a user logs on to a machine,
  - logon process collects credentials (e.g. user password) and presents them to the LSA,
  - LSA (*lsass.exe*) verifies the credentials,
  - logon process starts a shell (*explorer.exe*) in a new logon session under the user (principal).
- Shell spawns processes to the same logon session.
- Log-off destroys logon session and all processes in it.

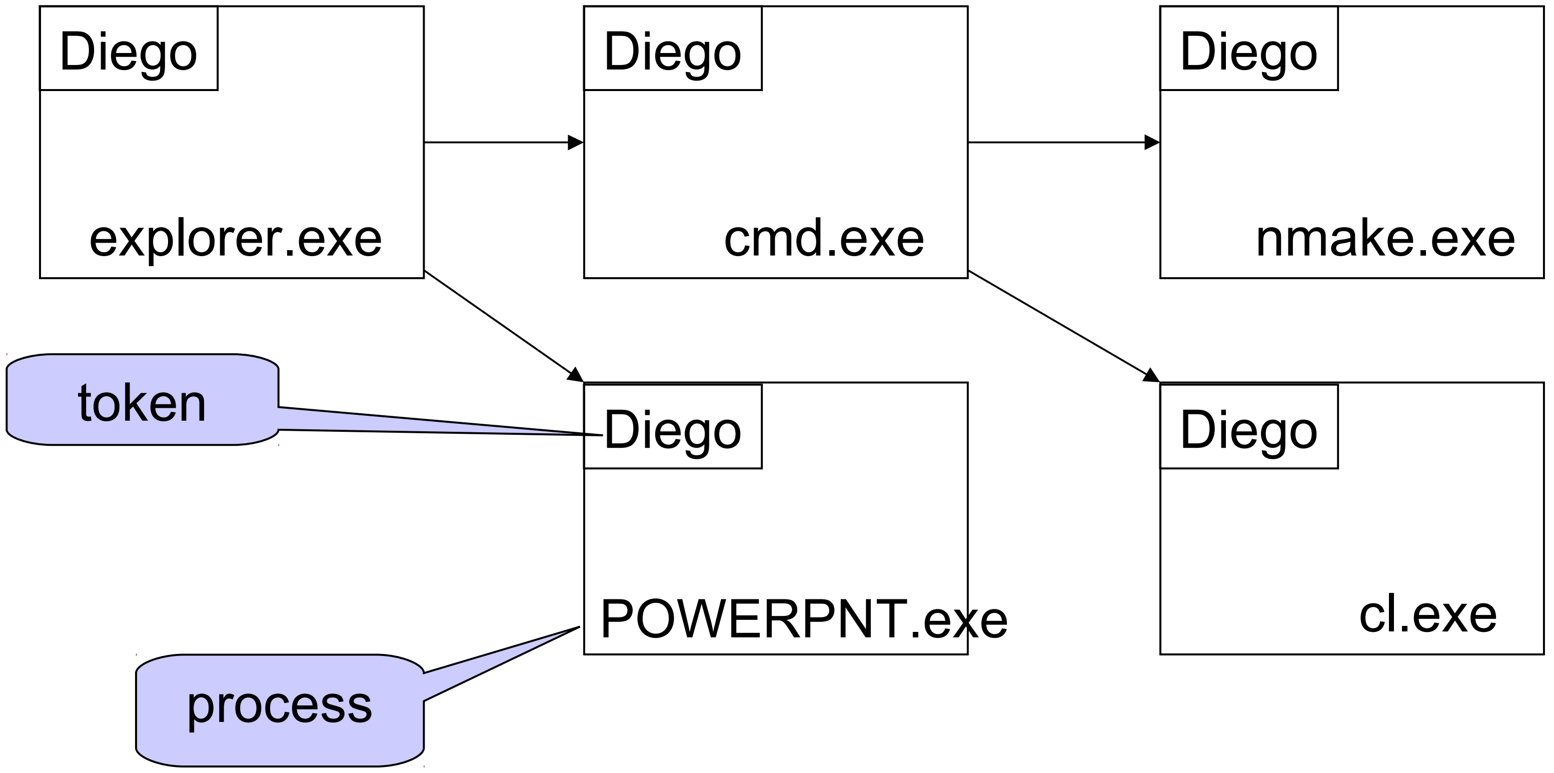
Authentication in Practice

- 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.

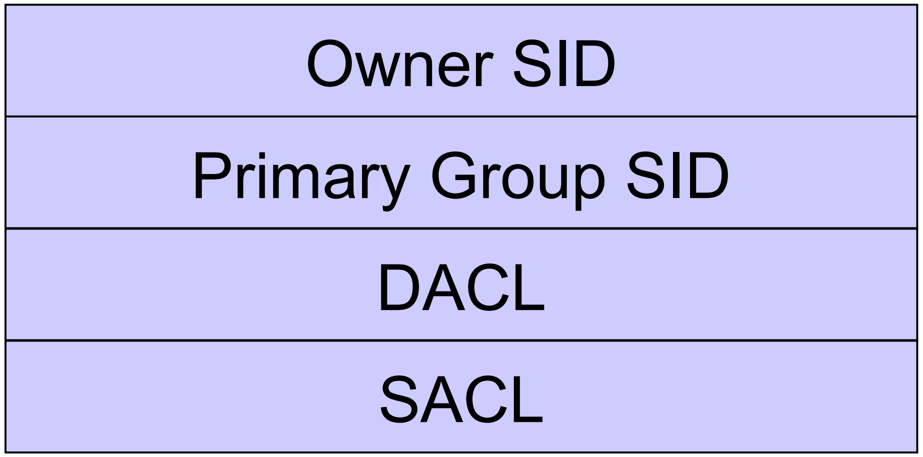
Processes in a Logon Session



Objects

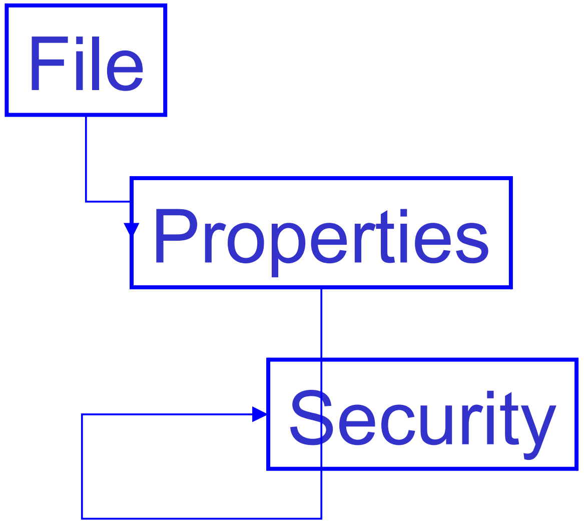
- Executive objects: processes, threads,…
- File system objects: files, directories.
- Registry keys, printers, …
- Securable objects have a security descriptor.
- Security descriptors for built-in objects are managed by the O/S.
- Security descriptors for private objects have to be managed by the application software.
  - Creating securable objects it is tedious but enables highly granular access control.

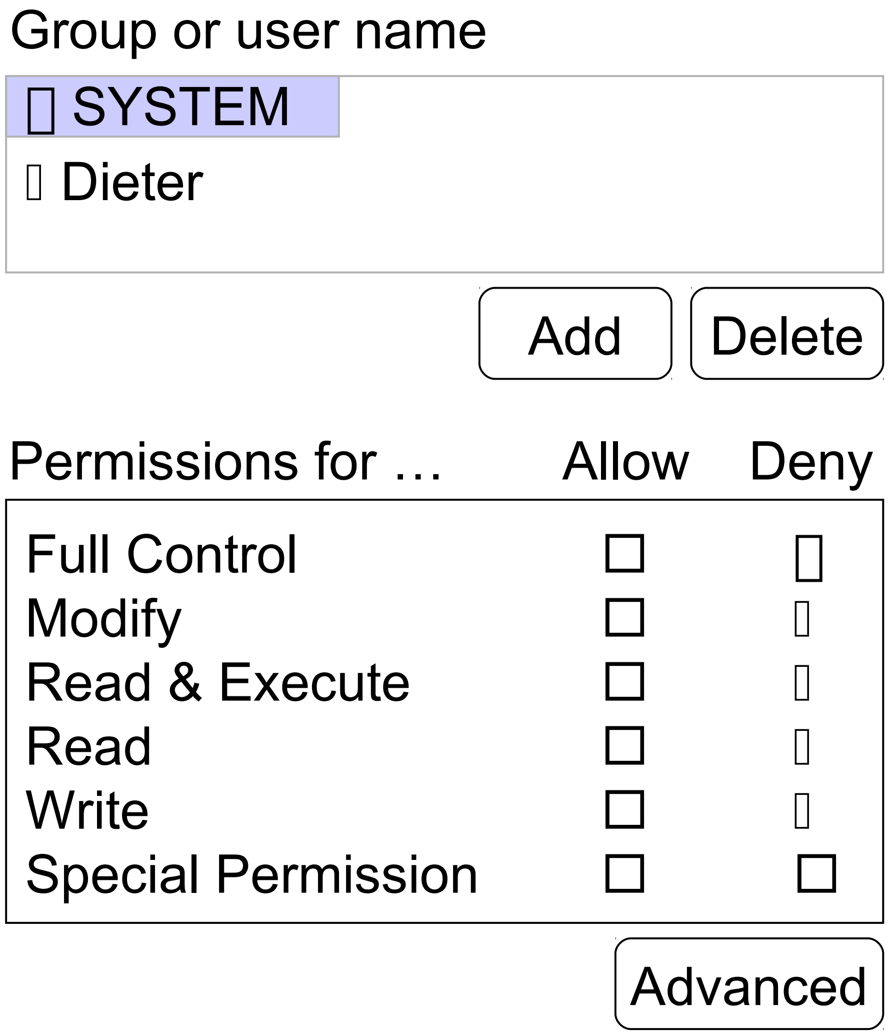
Security Descriptor (SD)



- Owner: defined when object is created.
- Primary Group: for POSIX compatibility
- DACL: lists who is granted or denied access
- SACL: defines audit policy

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

Owner

- Objects get an owner when they are created.
- The owner is a principal.
- Stored in the security descriptor the object.
- Owner (almost) 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 Lists

- DACL in security descriptor: List of access control entries (ACEs).
- ACE format:
  - Access mask (access rights)
  - Flags
  - Type: positive (Access Allowed), negative (Access Denied), audit; whether ACE contains ObjectType
  - InheritedObjectType (since Windows 2000)
  - ObjectType (since Windows 2000)
  - Trustee: Principal SID the ACE applies to

Permissions (access rights)

- Describe what one can do with an object.
- Permissions 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.

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.
- AD can be dynamically extended by adding new object types or new properties to existing object types.

ObjectType

- ObjectType: GUID defining an object type.
- ACEs without ObjectType are applied to all objects.
- Application can include ObjectType in its access requests; only ACEs with matching ObjectType or without an ObjectType will be evaluated.
  - Control read/write access on object property: put GUID of the property in ObjectType.
  - Control create/delete access on objects: put GUID of the object type in ObjectType.

Example

ACE1

Access mask:	create child
Type:	ACCESS_ALLOWED_OBJECT_ACE
InheritedObectType:	{GUID for RPC Services}
ObjectType	{GUID for RPC Endpoint}
Trustee (principal SID):	Server Applications

- 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 Control

- Access control decisions consider the
  - subject requesting access; credentials of the subject, including its principal, 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.
- Owner always has READ_CONTROL and WRITE_DAC permission (until Windows 7).

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 if a relevant negative ACE is found.
  - Deny access once the end of the DACL is reached (hence not all required permissions have been granted).
- For negative ACEs to take precedence over positive ACEs, they must be placed at the top of the DACL.
- To define "exceptions from exceptions" negative ACEs may be placed after positive ACEs.

Performance and Reliability

- Desired access 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 currently open file handles.
- Better performance.
- Better reliability because all access control checks are made in advance, before process starts a task (compare later with stack walking).

Null DACL

- There is a difference between a data structure that does not exist and a data structure that is empty.
- Empty DACL: nobody is granted any permission.
- No DACL (NULL DACL): everybody gets all access.

Access Control Approaches

- Several ways to do Windows access control (with increasing granularity and complexity).
- Impersonation: access control based on the principal requesting access; process ‘impersonates’ user SID of its token; coarse but simple to implement.
  - Typical O/S concept; does not work well at 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.

Least Privilege

Restricted Tokens

- So far access control has implicitly referred to users.
- We may in addition want to control what certain programs can do.
  - Roger Needham on Titan (1960s): In particular, it was possible to use the identity of a program as a parameter for access-control decisions as well as, or instead of, the identity of the user, a feature which Cambridge people have ever since regarded as strange to omit.
- It is usual but not particularly helpful to refer to such programs as "untrusted code".
- In Windows this issue could be addressed with restricted tokens.

Restricted Tokens – Theory

- 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.
- Process with a restricted token gets access only if SID and restricted SID are granted access.
- Restricted SIDs could be created
  - per program; SID has to be entered into the ACLs of the objects (object types) the program should have access to.
  - per object type and be added to restricted tokens depending on the program executed by the subject.

Restricted SID – Theory

Process with a restricted token gets access only if SID and restricted SID are granted access

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

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

Ace 2:
Access Rights: read
Principal SID: MyApp

read access granted

Restricted SID – Theory

Process with a restricted token gets access only if SID and restricted SID are granted access

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

Ace 1:
Access Rights: read
Principal SID: Admin

Ace 2:
Access Rights: read
Principal SID: MyApp

read access denied

Restricted SID – Theory

Process with a restricted token gets access only if SID and restricted SID are granted access

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

Ace 1:
Access Rights: read
Principal SID: Admin

Ace 2:
Access Rights: read
Principal SID: MyApp

read access denied

Restricted SID – Theory

Process with a restricted token gets access only if SID and restricted SID are granted access

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

Ace 1:
Access Rights: read
Principal SID: Diego

read access denied

User Account Control (UAC)

- Restricted tokens would – in theory – be an option to limit the access rights of a user depending on the application that is running.
- In practice, security policies may be hard to define:

  “This application might be dangerous. Do you want to restrict your privileges?”
- Vista implements only a limited version of restricted tokens; when a user in an administrator group logs in, two tokens (admin, user) are created.

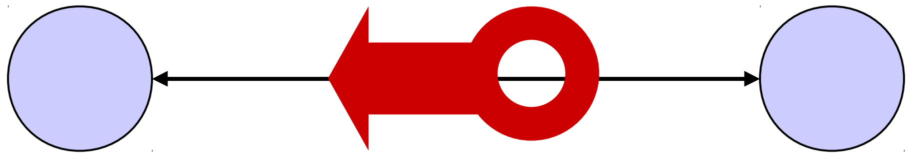
Default Accounts

- 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.

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.

Man-machine scale: Windows



specific complex focus on users	generic simple focus on data
man oriented	machine oriented

Windows 7 NT

Summary – General points

- Security identifiers tied to authorities; no side effects when policies of different authorities are merged.
- Generic policy defined at development time; specific policy instantiated at deployment time.
- Structuring policies and setting default values: inheritance of ACEs.
- Decision algorithm: traverse list until all required rights are collected; result depends on order of list entries.
- Securing private objects is the developer’s task.