Hochschule Konstanz
Fakultät Informatik

Introduction to IT Security

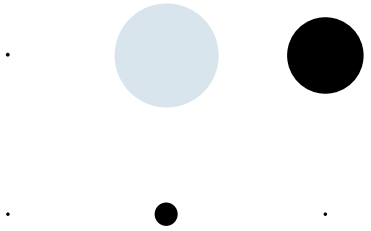
WIN+AIN

Hanno Langweg

04c Secure Operating Environments - Access Control

Secure Operating Environments

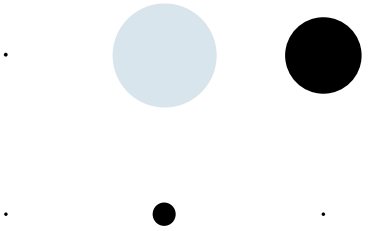
- Security of operating systems
- Trusted Computing
- Access control
- Malware



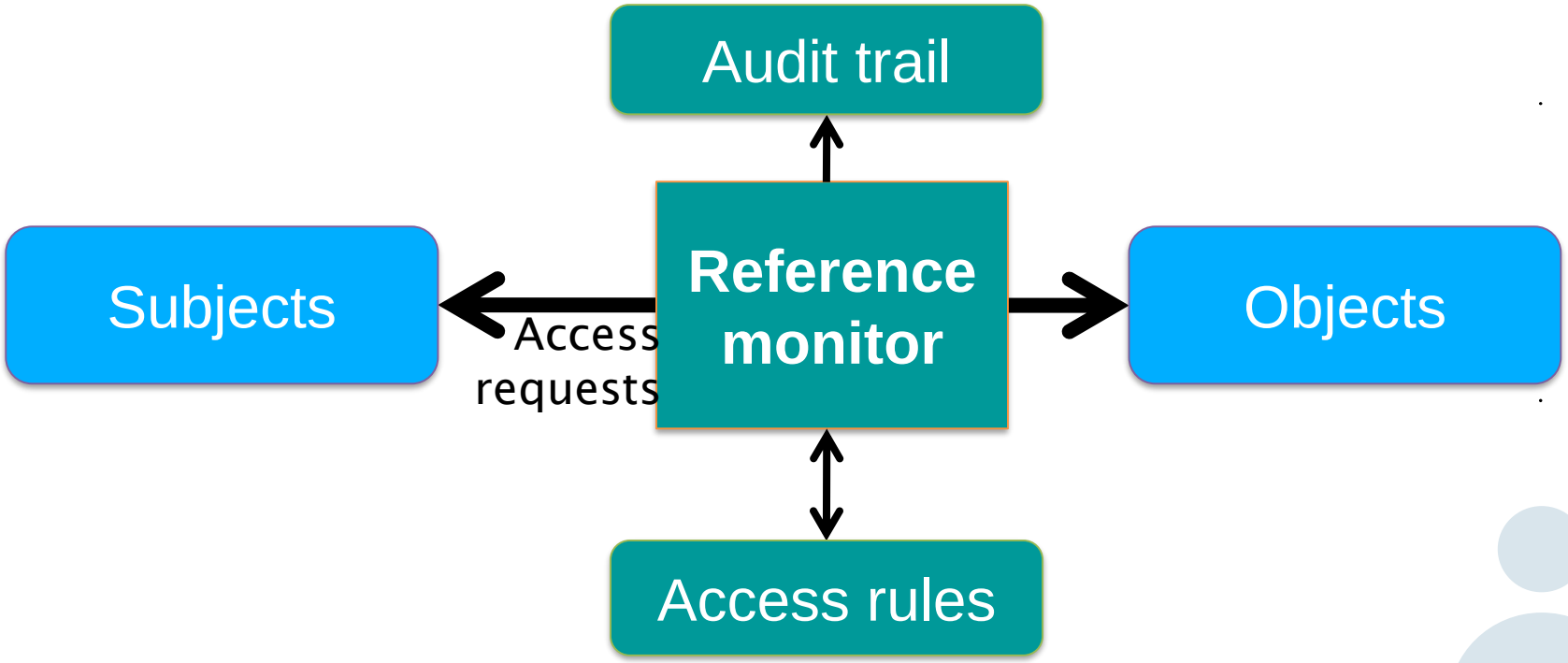
Access Control

Access control

- Access control = authentication + **authorisation**
  - Authentication: Verifying identity of subject
  - Authorisation: Verifying that subject has right to perform requested action on object
- Subjects request actions on objects
  - Alice wants to read a file
  - Bob wants to update account balance
- Process wants to open a socket



Reference monitor



Discretionary access control (DAC)

- **Data owners**, usually users, **set access rights**
- Subjects are trusted to make decisions
  - Users decide who is allowed to access their files
  - User or process that can read a secret file can also share it e.g. by email
- Typical in **commercial** and **consumer** systems
- There may be a policy against sharing and access may be audited, but the policy is not enforced technically
- Example of DAC outside computers:
  - Person with a key can open the door to others; door keys can be shared and copied

Access control list (ACL)

- **ACL = list of the access rights associated with an object**
  - file1.txt ACL:
    Alice: **{ read, write }**; Bob: **{ read }**;
    Process 4567: **{ read, write }**; Process 6789: **{append}.**
  - file2.txt ACL:
    Alice: **{ write }**; Bob: **{ read }.**
  - Socket s ACL:
    Process 6789: **{ open, read, write, close }.**
- ACL examples:
  - Windows/Unix file system

Capabilities

- **Capability = access right associated with the subject**
  - Alice's capabilities:
    file1.txt: **{ read, write }**; file2.txt: **{ write }**.
  - Bob's capabilities:
    file1.txt: **{ read }**; file2.txt: **{ read }**.
  - Process 4567 capabilities:
    file1.txt: **{ read, write }**.
  - Process 6789 capabilities:
    file1.txt: **{ append }**; Socket s: **{open, read, write, close }**.
- Examples of capabilities:
  - Mobile app privileges

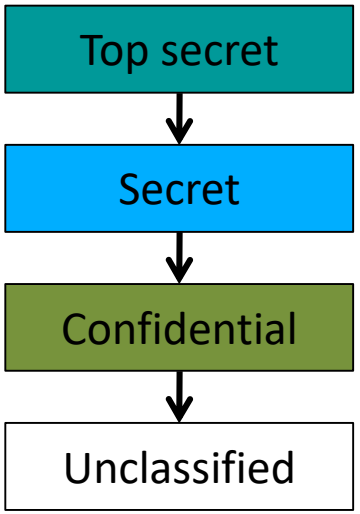
Mandatory access control (MAC)

- **Access rights based on rules (i.e. policy) set by administration**
- AC **policy enforced** and **cannot be changed by users**
- **Subjects cannot leak access rights to others**
  - User can read secret file, but cannot copy, print or email; file viewer prevents cut&paste and screen shots
  - One process can access the Internet, another writes files to disk, neither is allowed to do both
- MAC originates from **military policies**
  - Officer can read secret paper but cannot take copy out of room
  - Officer who has had contact with foreign agents may lose access to classified information

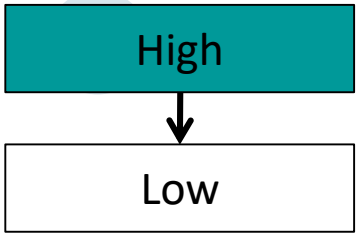
Clearance and classification

- Mandatory access control rules are often based on **security labels** on subjects and objects
  - Subject **clearance**
  - Object **classification**

    l : (Subjects ∪ Objects) → Labels
- MAC based on clearance and classification levels also called **multi-level security (MLS)**

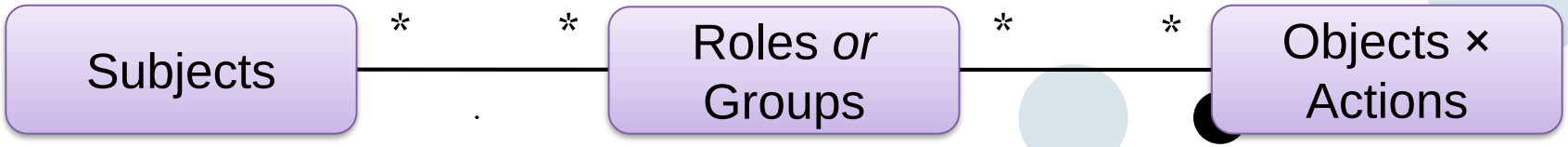


- **Simple security property:**
  S can read O if and only if $l(S) \geq l(O)$



Groups and roles

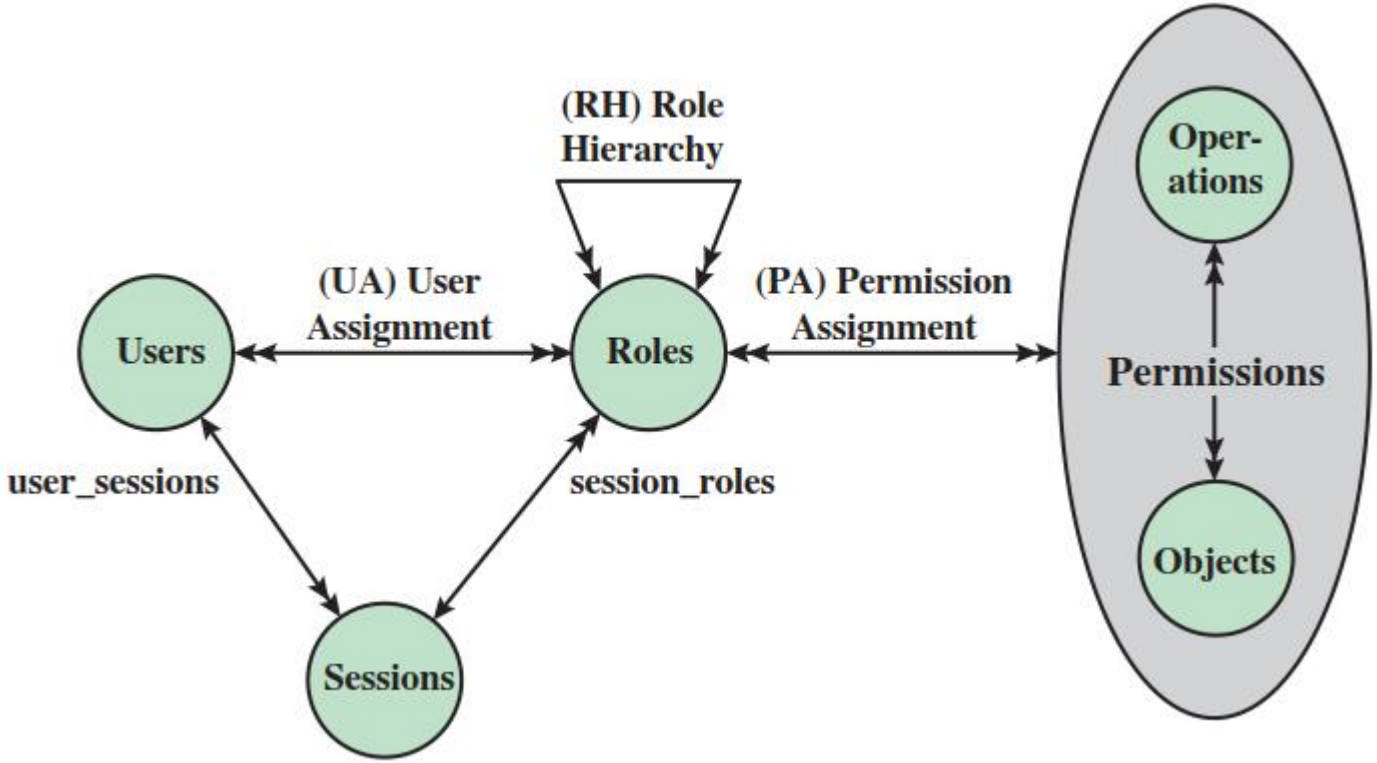
- Adding structure to policies
- **Group** = **set of subjects**
  - E.g. Administrators, CS students
  - Object ACL can list groups in addition to users
  - Both group membership and ACLs change over time
- **Role** = **set of permissions**
  (i.e. permitted actions on objects)
  - E.g. Administrator, INITSEC-teacher, IN-professor
  - Roles usually quite static; assignment to users changes



Role-based access control (RBAC)

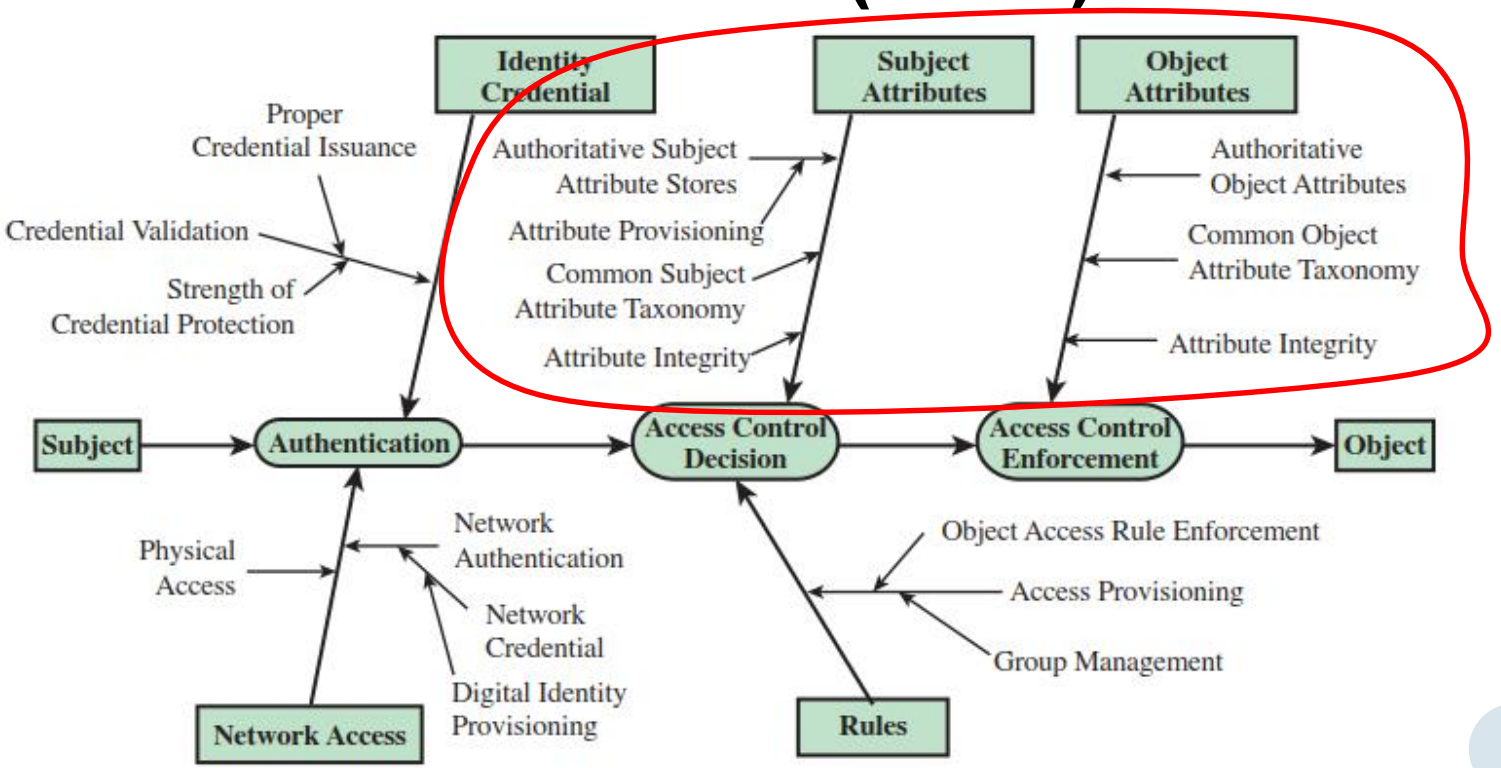
- Modeling **high-level roles** in an **organisation**
  - E.g. Doctor, Nurse, Student, Lecturer, Course-assistant
  - Roles defined once; changed infrequently
- Roles may be **parameterised**
  - E.g. Treating-doctor of Mr. Smith,
    Lecturer of INITSEC, Student of INITSEC
- Roles may form a **hierarchy** with inheritance
  - E.g. Lecturer and Teaching-assistant are Teaching-staff
- Roles are **assigned** to users for longer term but **activated** on demand for each **session**
- **Constraints** on role assignment and activation can implement separation of duty

Role-based access control (RBAC)



Stallings/Brown (2015). Computer Security. Figure 4.8

Attribute-based acc. cont. (ABAC)



- Base access decisions not just on subject identity, but also on subject/object/environment attributes, e.g. affiliation, type, time, place, content …

Stallings/Brown (2015). Computer Security. Figure 4.11

Discretionary Access Control (Microsoft Windows)

Windows Security Model

- **Principals** = users, machines, groups,…
- **Objects** = files, registry keys, printers, …
- Each object has a *discretionary access control list (DACL)*
- The active **subjects** are **processes** and **threads**
- Each process (or thread) has an **access token**
- When is a process allowed to access an object?
  - Object DACL compared with process's access token when creating handle to object

Windows objects security

- Securable objects
- Session
- SID
- Token
- Privileges
- Security descriptor

Securable objects in Windows

- **Object**: unit of **abstraction** for Windows system resources
- Instances of a type
- Instantiated by *Create*()* functions, referenced by a **handle**
- **Consistent interface** allows application of unified access control mechanisms

Common securable objects

- NTFS files/directories, network shares
- IPC objects (event, mutex, semaphore, pipe)
- Processes, threads, job objects, services
- Window stations, desktops (but not windows)
- Registry keys (but not registry values)
- Directory service (AD) objects
- Printers

Session

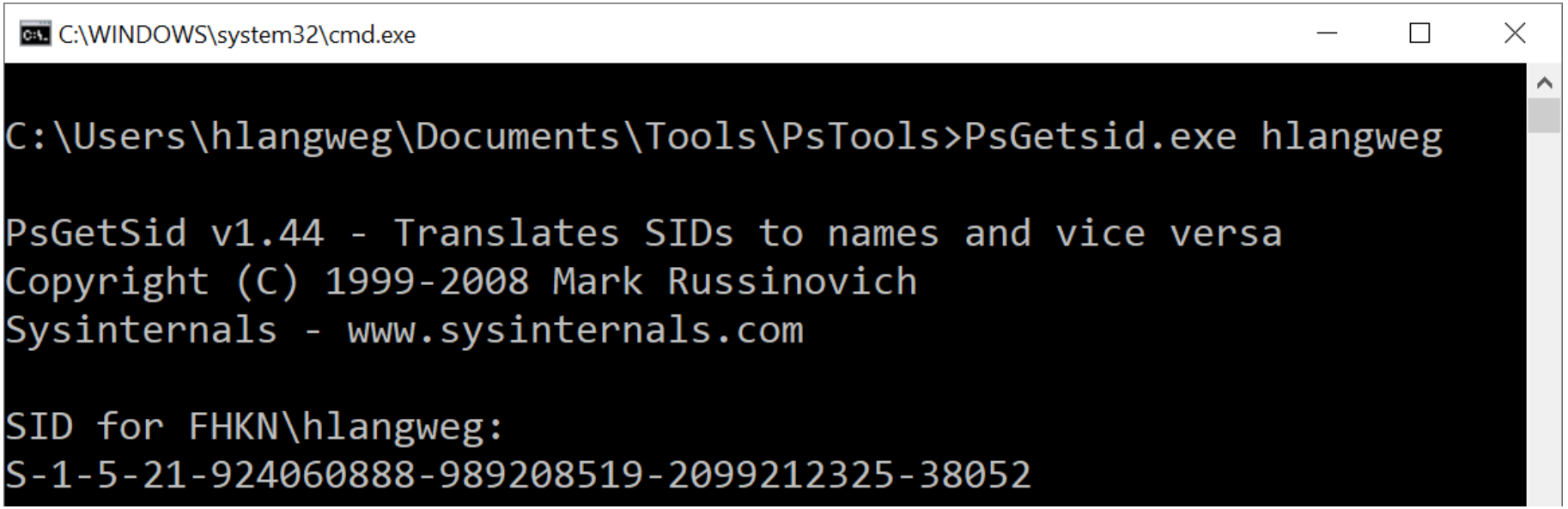
- Encapsulates data related to a logon instance
- Includes
  - Process access rights
  - Data accessible to processes
  - Behavioral characteristics for processes
- Isolates applications of logged-on users (from other users)

SID Security identifier

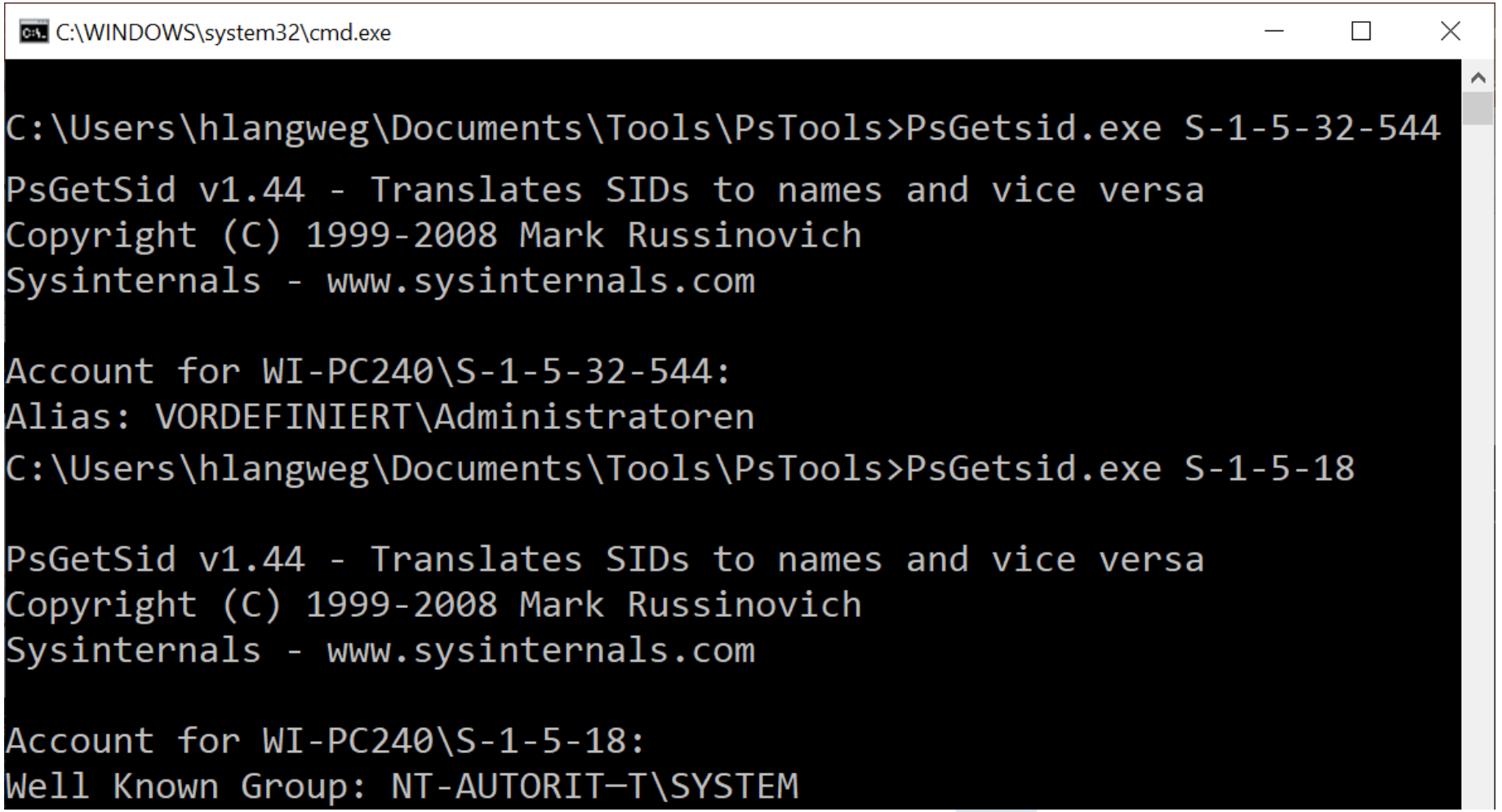
- **Unique identifier** for user/group/service/machine accounts
- SIDs do not change after assignment
- Format
  `S-<rev.id.>-<id. authority>-<sub.auth.>-<RID>`
- Well-known SIDs on every system:
  - `S-1-5-<domain id.>-500` Domain administrator
  - `S-1-5-32-544` *Administrators* group
  - `S-1-1-0` *Everyone* group
  - `S-1-5-18` *Local system* account

SID examples

- Download PsTools, use `PsGetSid.exe`
  https://docs.microsoft.com/de-de/sysinternals/downloads/pstools



SID examples



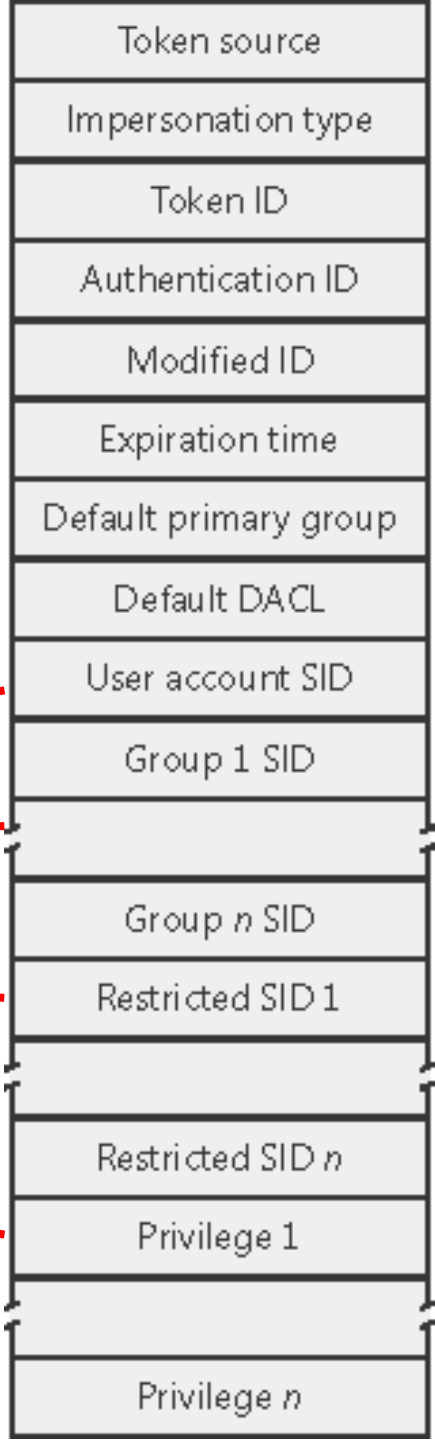
Token

- Describes **security context** for process/thread
- **Created** when new session is started
  - **Inherited** by child processes
- Create different token
  - LogonUser(), CreateProcessAsUser()
  - CreateProcessWithLogon() (used by RunAs service)

- Weekend video suggestion:
  Raiders of the Elevated Token: Understanding User Account Control and App Capabilities in Windows 8
  https://channel9.msdn.com/Events/TechEd/NorthAmerica/2013/WCA-B335#fbid=

Token

- Contains
  - Logon session identifier (as a group SID)
  - Default DACL used when no DACL specified for object creation
  - SID for user, SIDs for groups user belongs to
  - Restricting SID list, SIDs that must not be used to get access
  - Privilege list

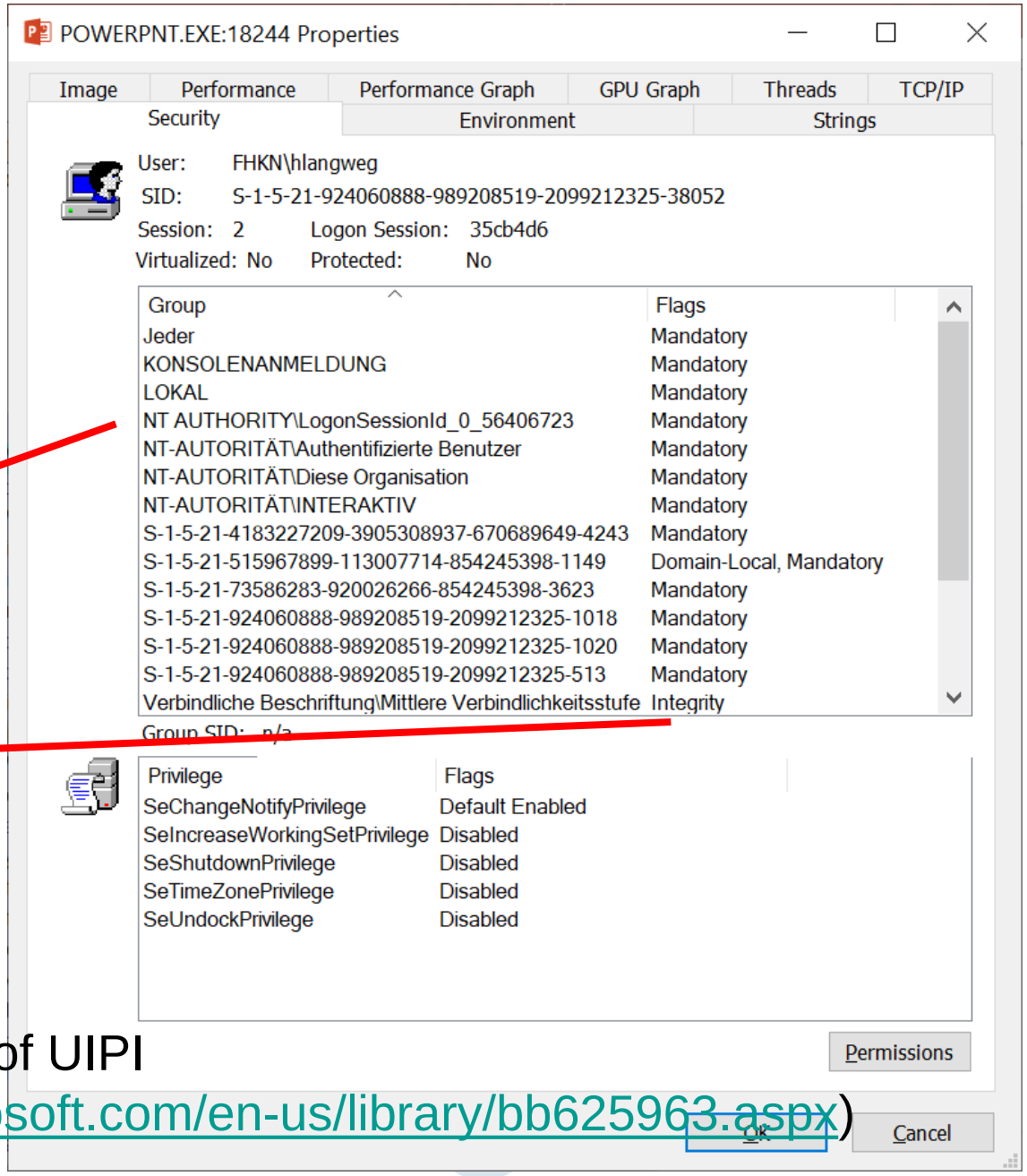


Restricted tokens

- Access token inherited by a process may give it too many access rights
- Process may create a **restricted token**
  - *remove privileges*
  - *disable groups*: change SIDs to deny-only groups, which are not deleted but marked as USE_FOR_DENY_ONLY
  - *add restricted SIDs*: a second list of SIDs that is also compared against DACLs
- Process can assign restricted tokens to its child processes or threads
- Typically used in services, rarely in desktop apps

ProcessExplorer

- Note:
  - Logon SID as group SID
  - Integrity level as group SID
  - (Implementation of UIPI http://msdn.microsoft.com/en-us/library/bb625963.aspx)



```
typedef enum _TOKEN_INFORMATION_CLASS {
  TokenUser                                = 1,
  TokenGroups,
  TokenPrivileges,
  TokenOwner,
  TokenPrimaryGroup,
  TokenDefaultDacl,
  TokenSource,
  TokenType,
  TokenImpersonationLevel,
  TokenStatistics,
  TokenRestrictedSids,
  TokenSessionId,
  TokenGroupsAndPrivileges,
  TokenSessionReference,
  TokenSandBoxInert,
  TokenAuditPolicy,
  TokenOrigin,
  TokenElevationType,
  TokenLinkedToken,
  TokenElevation,
  TokenHasRestrictions,
  TokenAccessInformation,
  TokenVirtualizationAllowed,
  TokenVirtualizationEnabled,
  TokenIntegrityLevel,
  TokenUIAccess,
  TokenMandatoryPolicy,
  TokenLogonSid,
  TokenIsAppContainer,
  TokenCapabilities,
  TokenAppContainerSid,
  TokenAppContainerNumber,
  TokenUserClaimAttributes,
  TokenDeviceClaimAttributes,
  TokenRestrictedUserClaimAttributes,
  TokenRestrictedDeviceClaimAttributes,
  TokenDeviceGroups,
  TokenRestrictedDeviceGroups,
  TokenSecurityAttributes,
  TokenIsRestricted,
  MaxTokenInfoClass
} TOKEN_INFORMATION_CLASS, *PTOKEN_INFORMATION_CLASS;
```

winnt.h

Privileges

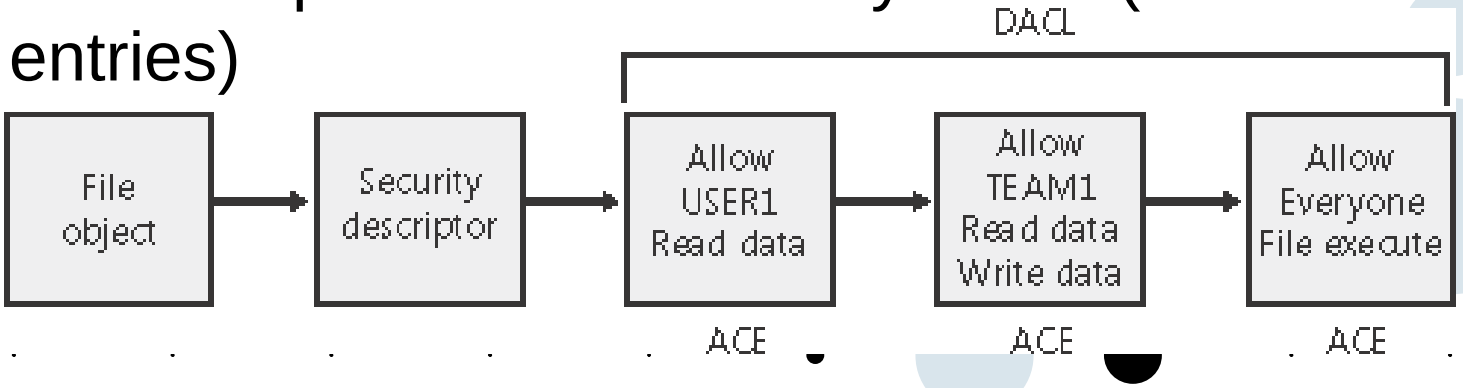
- Special permissions for **system-related** tasks, used where ACLs not suitable
- Set via group policy
- Included in token
- Examples
  - SeDebugPrivilege - allows debugging of processes
  - SeLoadDriverPrivilege - allows loading of device drivers
  - SeTakeOwnershipPrivilege - allows to take ownership of other users' objects and files

```
#define SE_CREATE_TOKEN_NAME             TEXT("SeCreateTokenPrivilege")
#define SE_ASSIGNPRIMARYTOKEN_NAME       TEXT("SeAssignPrimaryTokenPrivilege")
#define SE_LOCK_MEMORY_NAME              TEXT("SeLockMemoryPrivilege")
#define SE_INCREASE_QUOTA_NAME           TEXT("SeIncreaseQuotaPrivilege")
#define SE_UNSOLICITED_INPUT_NAME        TEXT("SeUnsolicitedInputPrivilege")
#define SE_MACHINE_ACCOUNT_NAME          TEXT("SeMachineAccountPrivilege")
#define SE_TCB_NAME                      TEXT("SeTcbPrivilege")
#define SE_SECURITY_NAME                 TEXT("SeSecurityPrivilege")
#define SE_TAKE_OWNERSHIP_NAME           TEXT("SeTakeOwnershipPrivilege")
#define SE_LOAD_DRIVER_NAME              TEXT("SeLoadDriverPrivilege")
#define SE_SYSTEM_PROFILE_NAME           TEXT("SeSystemProfilePrivilege")
#define SE_SYSTEMTIME_NAME               TEXT("SeSystemtimePrivilege")
#define SE_PROF_SINGLE_PROCESS_NAME      TEXT("SeProfileSingleProcessPrivilege")
#define SE_INC_BASE_PRIORITY_NAME        TEXT("SeIncreaseBasePriorityPrivilege")
#define SE_CREATE_PAGEFILE_NAME          TEXT("SeCreatePagefilePrivilege")
#define SE_CREATE_PERMANENT_NAME         TEXT("SeCreatePermanentPrivilege")
#define SE_BACKUP_NAME                   TEXT("SeBackupPrivilege")
#define SE_RESTORE_NAME                  TEXT("SeRestorePrivilege")
#define SE_SHUTDOWN_NAME                 TEXT("SeShutdownPrivilege")
#define SE_DEBUG_NAME                    TEXT("SeDebugPrivilege")
#define SE_AUDIT_NAME                    TEXT("SeAuditPrivilege")
#define SE_SYSTEM_ENVIRONMENT_NAME       TEXT("SeSystemEnvironmentPrivilege")
#define SE_CHANGE_NOTIFY_NAME            TEXT("SeChangeNotifyPrivilege")
#define SE_REMOTE_SHUTDOWN_NAME          TEXT("SeRemoteShutdownPrivilege")
#define SE_UNDOCK_NAME                   TEXT("SeUndockPrivilege")
#define SE_SYNC_AGENT_NAME               TEXT("SeSyncAgentPrivilege")
#define SE_ENABLE_DELEGATION_NAME        TEXT("SeEnableDelegationPrivilege")
#define SE_MANAGE_VOLUME_NAME            TEXT("SeManageVolumePrivilege")
#define SE_IMPERSONATE_NAME              TEXT("SeImpersonatePrivilege")
#define SE_CREATE_GLOBAL_NAME            TEXT("SeCreateGlobalPrivilege")
#define SE_TRUSTED_CREDMAN_ACCESS_NAME   TEXT("SeTrustedCredManAccessPrivilege")
#define SE_RELABEL_NAME                  TEXT("SeRelabelPrivilege")
#define SE_INC_WORKING_SET_NAME          TEXT("SeIncreaseWorkingSetPrivilege")
#define SE_TIME_ZONE_NAME                TEXT("SeTimeZonePrivilege")
#define SE_CREATE_SYMBOLIC_LINK_NAME     TEXT("SeCreateSymbolicLinkPrivilege")
```

winnt.h

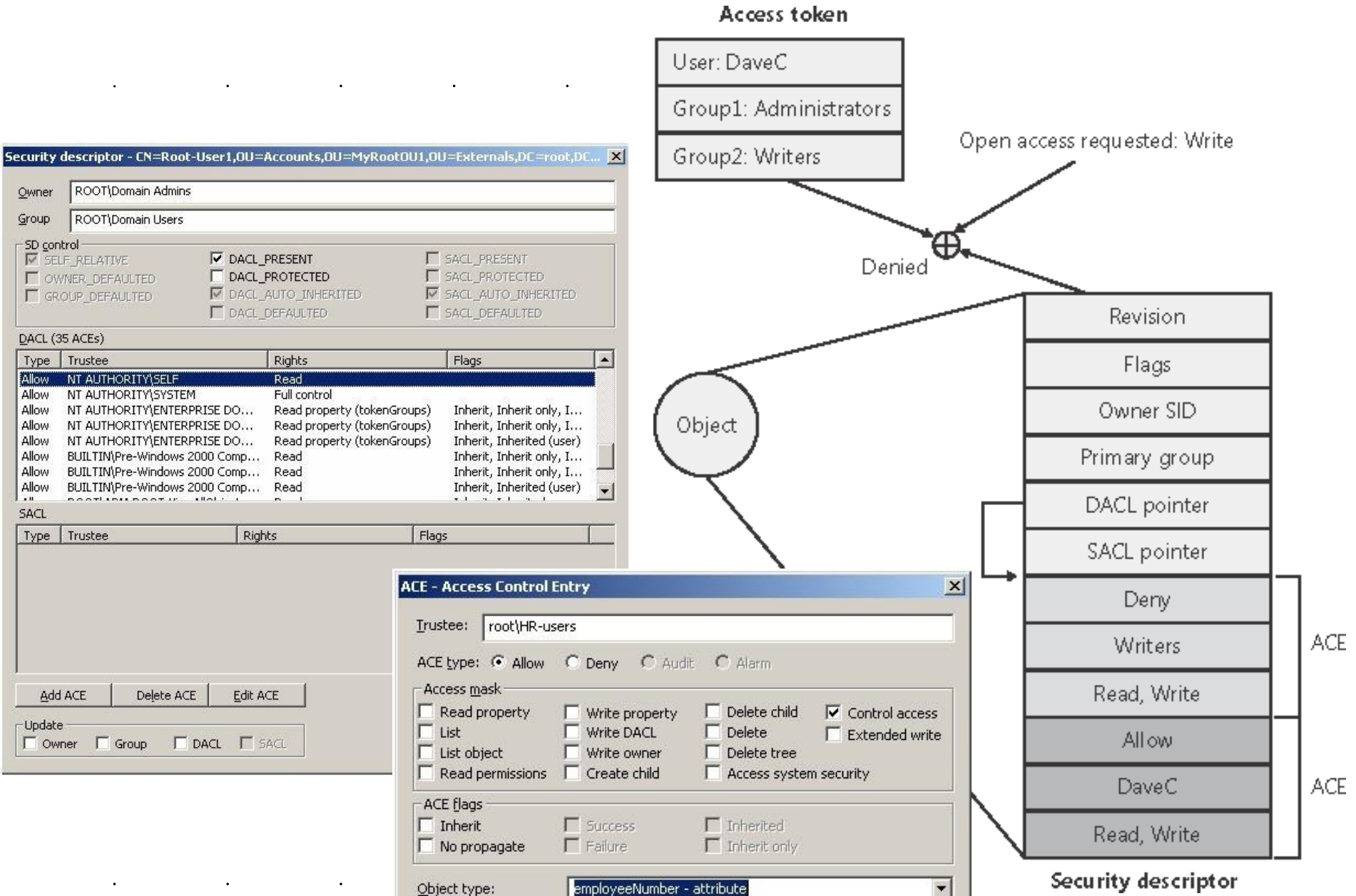
Security descriptor

- Description of securable objects
  - Owner SID - owning user/group
  - Group SID - owning group (mostly unused)
  - DACL - accounts SIDs and access permissions
  - SACL - groups+accessess that trigger logging event
- ACL composed of allow/deny ACEs (AC entries)



Permissions

- Permissions are actions that apply to each object class
- Some **generic permissions** are defined for all objects: *read*, *write*, *execute*, *all*, *delete*, etc.
- **Specific permissions** are defined for each object class: Append, AddSubDir, CreateThread,etc.
- Permissions are encoded as a 32-bit mask
- Object DACL specifies which principals (SIDs) have which permissions


Access token
User: DaveC
Group1: Administrators
Group2: Writers
Open access requested: Write
Denied
Object
Revision
Flags
Owner SID
Primary group
DACL pointer
SACL pointer
Deny
Writers
Read, Write
Allow
DaveC
Read, Write
ACE
ACE
Security descriptor
Security descriptor - CN=Root-User1,OU=Accounts,OU=MyRootOU1,OU=Externals,DC=root,DC...
Owner
ROOT\Domain Admins
Group
ROOT\Domain Users
SD control
SELF_RELATIVE
OWNER_DEFAULTED
GROUP_DEFAULTED
DACL_PRESENT
DACL_PROTECTED
DACL_AUTO_INHERITED
DACL_DEFAULTED
SACL_PRESENT
SACL_PROTECTED
SACL_AUTO_INHERITED
SACL_DEFAULTED
DACL (35 ACEs)
Type
Trustee
Rights
Flags
Allow NT AUTHORITY\SELF Read
Allow NT AUTHORITY\SYSTEM Full control
Allow NT AUTHORITY\ENTERPRISE DO... Read property (tokenGroups) Inherit, Inherit only, I...
Allow NT AUTHORITY\ENTERPRISE DO... Read property (tokenGroups) Inherit, Inherit only, I...
Allow NT AUTHORITY\ENTERPRISE DO... Read property (tokenGroups) Inherit, Inherited (user)
Allow BUILTIN\Pre-Windows 2000 Comp... Read Inherit, Inherit only, I...
Allow BUILTIN\Pre-Windows 2000 Comp... Read Inherit, Inherit only, I...
Allow BUILTIN\Pre-Windows 2000 Comp... Read Inherit, Inherited (user)
SACL
Type
Trustee
Rights
Flags
Add ACE
Delete ACE
Edit ACE
Update
Owner
Group
DACL
SACL
ACE - Access Control Entry
Trustee:
root\HR-users
ACE type:
Allow
Deny
Audit
Alarm
Access mask
Read property
List
List object
Read permissions
Write property
Write DACL
Write owner
Create child
Delete child
Delete
Delete tree
Access system security
Control access
Extended write
ACE flags
Inherit
No propagate
Success
Failure
Inherited
Inherit only
Object type:
employeeNumber - attribute
Inherited object type:
(none)
OK
Cancel

LDP.exe

Review of ACL use

- *NULL* DACL - everyone allowed
- *Empty* DACL - no one allowed
- ACL inheritance (not discussed here)
- ACE order
  - Comparison of SIDs and access masks
  - Evaluation completed when match found
  - Deny ACEs need to be first in list to take precedence
  - Evaluation ends with denied access if no match found
- Further reading: Access Check Algorithm Pseudocode http://msdn.microsoft.com/en-us/library/cc230290.aspx

Performance and reliability

- Group membership and privileges determined at login time
  - User's group SIDs cached in token of login process; sub-processes get a copy
  - Token will not change even if a membership or privilege is revoked from a SID
- Desired access is compared against token and DACL when creating handle to an object – not at access time
  - Changing file DACL does not affect open file handles
- Consequences:
  - Better performance because of fewer checks
  - Better reliability because a process knows in advance whether it has sufficient access rights for a task
  - No immediate revocation of access rights

Discretionary Access Control (Unix)

Principals

- The principals are **users** and **groups**
- Users have username and user identifier (UID)
- Groups have **group name**, group identifier (GID)
- UID and GID are usually 16-bit numbers, e.g.

  0 = root

  19042 = hlangweg

  100 = users
- Both names and identifiers difficult to change once selected
  - References to home directories
  - UID values often differ from system to system

User accounts

- User accounts are stored in `/etc/passwd`
  Format: username:password:UID:GID:name:homedir:shell
- Example:

```
root:x:0:0:root:/root:/bin/bash
mail:x:8:12:mail:/var/spool/mail:
ace:x:500:103:Alice:/home/ace:/bin/bash
carol:x:501:102:Carol:/home/carol:/bin/nologin
```

- Password hashes are stored in `/etc/shadow`
  (used to be in `/etc/passwd`)
  Format: username:hash:lastchange:daysuntilchangepermitted:
  daysuntilchangequired:dayswarningperiod:daysuntilinactive:
  expirydate:RFU

```
root:7kSSI2k.Df:18442:0:99999:7:::
mail:*:8:12:mail:/var/spool/mail:
ace:69geDfelkw:18442:0:42:7:::
carol:7fkKded:501:18442:0:99999:7:::
```

Superuser

- The superuser is a special privileged principal with UID zero and usually the user name root
- There are few restrictions on the superuser
  - All security checks are turned off for the superuser
  - The superuser can become any other user
- Examples:
  - The superuser cannot write to a read-only file system but can remount it as writeable
  - The superuser cannot decrypt passwords (because they are hash values) but can reset them

Groups

- Users belong to one or more groups
- The file `/etc/group` contains a list of all groups; file entry format:

  groupname:password:GID:list of users
- Example:

  `initsec:*:209:carol,al`
- Every user belongs to a primary group; the group ID (GID) of the primary group is stored in `/etc/passwd`
- Depending on the Unix OS, user can belong to only one or many groups at the same time
- Usually only superuser can add groups and members
- Use the **`groups`** command to see your groups

Subjects

- The **subjects** in Unix are processes; a process has a process ID (PID)
- Processes can create new processes
- Processes have a real UID and an effective UID (similarly for GID)
- **Real** UID/GID: inherited from the parent; typically UID/GID of the *user logged in*
- **Effective** UID/GID: inherited from the parent process or *from the file being executed*

Example

Process	UID real	UID effective	GID real	GID effective
/bin/login	**root**	**root**	**system**	**system**

User hlangweg logs on; the login process verifies the password and (with its superuser rights) changes its UID and GID (setuid(2), setguid(2)):

/bin/login hlangweg hlangweg prof prof

The login process executes the user's login shell:

/bin/bash hlangweg hlangweg prof prof

From the shell, the user executes a command, e.g. **ls**

/bin/ls hlangweg hlangweg prof prof

The user executes command **passwd** to change his password:

/bin/passwd hlangweg root prof system

Objects

- The **objects** of access control are **files**, **directories** and **devices**
  - Organised in a tree-structured file system
- Directory is a file containing file names and pointers to *inode* data structures
- **Inode** stores information about the object owner's user and group, and permissions

Information about objects

- Example: directory listing with **`ls -l`**

```
-rw-r--r-- 1 hlangweg prof 1617 Oct 28 11:01 my.tex
drwx------ 2 hlangweg prof  512 Oct 25 17:44 vl/
```

- *File type*: first character
  - '-' file
  - 'd' directory
  - 'b' block device file
  - 'c' character device file
  - 's' socket
  - 'l' symbolic link
  - 'p' FIFO pipe
- *File permissions*: nine characters
- *Link counter*: the number of links (i.e. directory entries pointing) to the inode

Information about objects

```
-rw-r--r-- 1 hlangweg prof 1617 Oct 28 11:01 my.tex
drwx------ 2 hlangweg prof  512 Oct 25 17:44 vl/
```

- *Username* of the owner: usually the user that has created the file
- *Group*: a newly created file usually belongs to its creator's primary group
- File size, modification time, filename
- Owner and root can change permissions (**chmod**); root can change the file owner and group (**chown**)
- User can change the file group to of its own groups
- *Filename is stored in the directory, not in inode*

File permissions

- Permission bits are grouped in three triples that define read, write, and execute access for **owner**, **group**, and **other**

- `rw-r--r--` read and write access for the owner, read access for group and other
- `rwx------` read, write, and execute access for the owner, no rights to group and other

File permissions

- SUID programs run with the effective UID of the owner of the executable file
- When **`ls -l`** displays a **SUID program**, the execute permission of the owner is given as **`s`** instead of **`x`**:

```
-rws--x–x 3 root bin 16384 Nov 16 1996 passwd*
```

- SGID programs run with the effective GID of the owner of the executable file
- When **`ls -l`** displays a **SGID program**, the execute permission of the group is given as **`s`** instead of **`x`**

Octal representation

- File permissions can also be specified as **octal numbers**
- Examples: `rw-r--r--` is equivalent to 644; `rwxrwxrwx` is equivalent to 777
- Conversion table:

0040 read by group	4000 set UID on execution
0020 write by group	2000 set GID on execution
0010 execute by group	1000 set sticky bit
0004 read by other	0400 read by owner
0002 write by other	0200 write by owner
0001 execute by other	0100 execute by owner

Access control decisions

- Access control uses the effective UID/GID:
  - If the subject's UID owns the file, the permission bits for owner decide whether access is granted
  - If the subject's UID does not own the file but its GID does, the permission bits for group decide whether access is granted
  - If the subject's UID and GID do not own the file, the permission bits for other (also called world) decide whether access is granted
- Note that although the permission bits may give the owner less access than to others, the owner can always change the permissions (*discretionary access control*)

Permissions for directories

- **Read** permission: to find which files are in the directory, e.g. for executing ls
- **Write** permission: to add files and delete files
- **Execute** permission: to make the directory the current directory (cd) and for opening files inside the directory
- E.g. every user has a home directory; what are the correct permissions for the home directory?

Default permissions

- Unix utilities typically use default permissions 666 for a new data file and 777 for a new executable file
- Permissions can be restricted with **umask**: a three-digit octal number specifying the rights that should be withheld

  File permissions = default AND (NOT umask)
- Sensible umask values:
  - 022: all permissions for the owner, read and execute permission for group and other
  - 037: all permissions for the owner, read permission for group, no permissions for other
  - 077: all permissions for the owner, no permissions for group and other
- Example: default permissions 666, umask 077 → permissions for new file 0600

Unix access control — discussion

- Unix permissions have been standardised by IEEE as part of the POSIX standards (DOI 10.1109/IEEESTD.1992.106983)
  - Fairly universal across Unix systems
- Limitations and advantages?
  - Files have only one owner and group
  - Complex policies, e.g. access to several groups, are impractical to implement
  - Superuser needed to maintain groups
  - All access rights (e.g. shutdown, create user) must be mapped to **file** access and to **read**, **write** and **execute** permissions
  - Relatively simple and widely understood
  - Relatively easy to check the protection state
- Unix versions have subtle differences and may implement additional access control features