Unit 6: Protection and Security

6.3. Windows 2000 Security Concepts

Windows 2000 Security Concepts

- Configurable set of security services
- U.S. DoD C2 level for trusted operating systems
- United Stated Nation Computer Security Center
  - 1995: stand-alone configurations of NT Server & NT Workstation 3.5 formally certified (see http://www.radium.ncsc.mil/tpep/epl)
- UK Information Security Evaluation and Certification
  - 1996: NT S/NT WS 3.51 stand-alone/networked configurations certified at F-C2/E3 level (see http://www.itsec.gov.uk)
- Windows NT 4.0 SP 6a has been certified for C2
  - In both, networked and stand-alone configurations (U.S. DoD NCSC and UK ITSEC)
- Windows 2000 is currently under evaluation

Windows 2000 Security Features

- User Accounts
- Passwords
- File and Directory Protection
- Registry Protection
- Printer Protection
- Auditing
- Performance Monitoring

Windows 2000 Security Features (contd.)

- Secure logon facility
  - Logon identifier & password
- Discretionary access control
  - Owner of a resource may control access rights
- Security auditing
  - Creation, access, deletion system resources
- Memory protection
  - Private virtual address spaces
  - Memory pages allocated to user processes will be zero-ed
- Windows 2000 enhancements:
  - Active Directory account management for distributed environments
  - Kerberos v.5, Secure Socket Layer 3.0, CryptoAPI, encrypting NTFS

Additional Resources

- Windows NT/2000 Srv. Concepts and Planning Manual
  (\support\books on NT Server CD, MSDN Lib, TechNet)
  - Working with User and Group Accounts
  - Managing User Work Environments
  - Managing Shared Resources and Resource Security
  - Monitoring Events
- Windows NT/2000 Workstation Resource Guide
  - Security / Security in a Software Development Environment chapters
- Platform Software Development Kit (SDK)
- Windows 2000 Device Driver Kit (DDK)
  - Kernel-mode interface to security functions

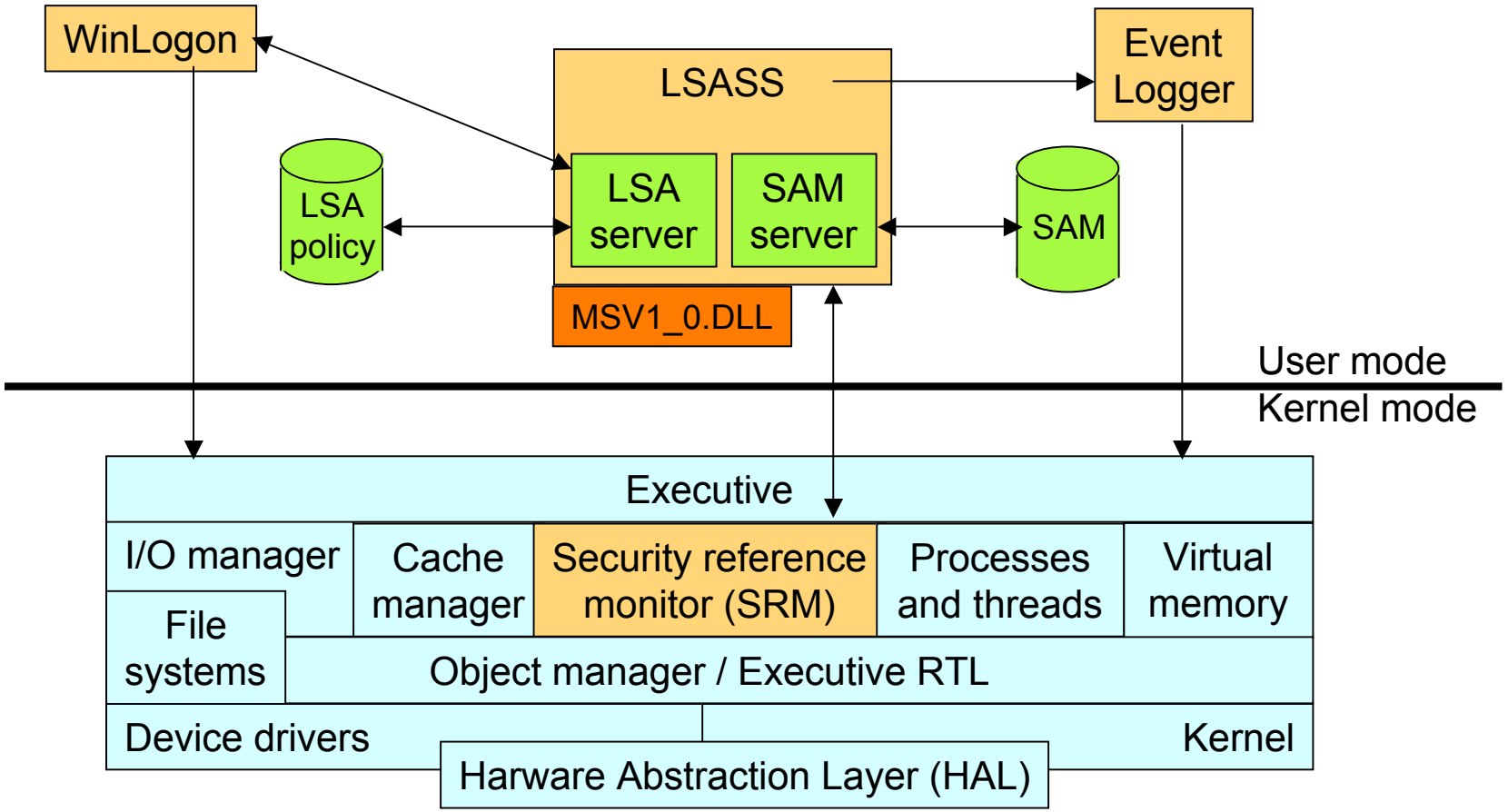
Security System Components

- Security reference monitor (SRM)
  - Component in NT executive (NTOSKRNL.EXE)
  - Security access checks on objects, privilege manipulation, auditing
- Local security authority (LSA) server
  - User-mode server (LSASS.EXE)
  - Local system security policy: passwd, users, groups, auditing settings
  - User authentication, sends security audit messages to Event Log
- LSA policy database
  - In registry at HKEY_LOCAL_MACHINE\Security
  - Contains: trusted domains, access permissions, privileges, auditing level
- Security accounts manager (SAM) server
  - Set of subroutines to manage users/groups database; in LSASS process

Security System Components (contd.)

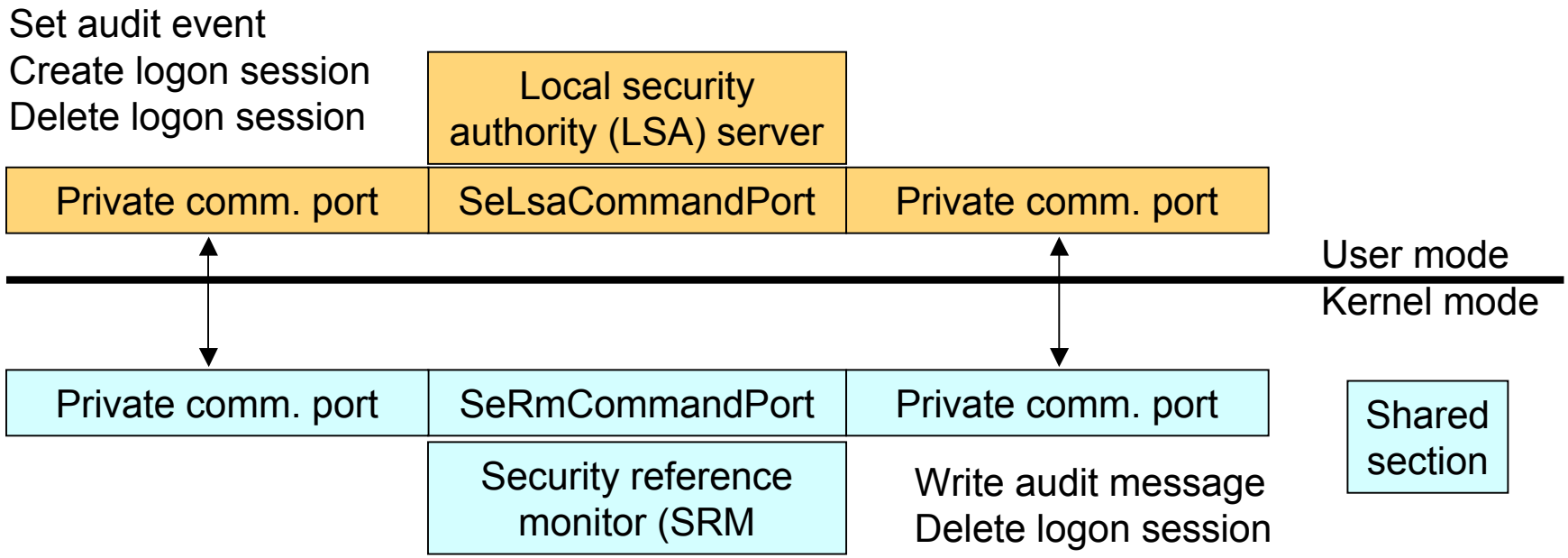
- SAM database
  - Users, groups, passwords: in registry at HKLM\SAM
- Default authentication package
  - MSV1_0.DLL, runs in context of LSASS process
  - Checks username/passwd against SAM entries; returns access token
- Logon process
  - User-mode process, WINLOGON.EXE
  - Sends username/passwd to LSA, creates initial process in user session
- Network logon service
  - User-mode service inside SERVICES.EXE
  - Responds to network logon requests, interacts with LSASS process

Security System Diagram



Communication between SRM and LSA

- Communication via local procedure call (LPC)
  - SeLsaCommandPort/SeRmCommand port for initialization
  - Usage of private ports/shared memory when initialization is completed

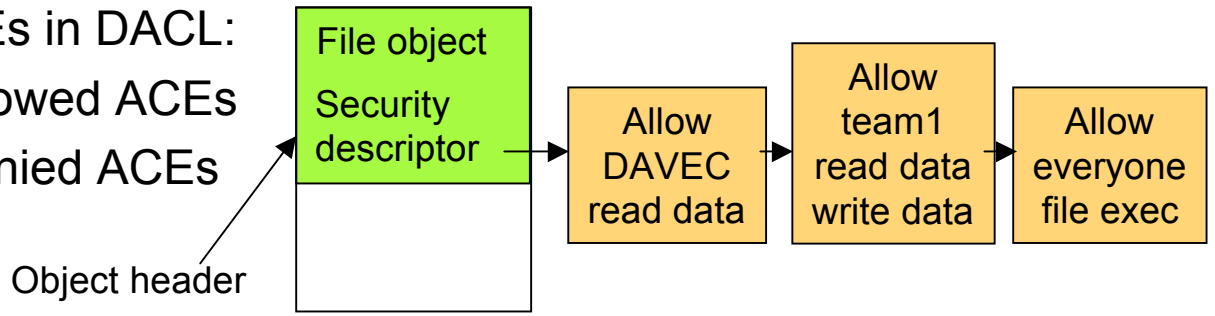


Security Descriptors and Access Control

- All securable objects are assigned security descriptors
- Attributes:
  - **Owner SID**: owners security ID
  - **Group SID**: the security ID of the primary group for the object (used only by POSIX)
  - **Discretionary access control list (DACL)**: specifies who has what access to the object
  - **System access control list (SACL)**: specifies which operations by which users should be logged in the security audit log
- Access Control List (ACL)
  - Header + Access Control Entries (ACEs)
  - ACL with zero ACEs (null ACE): no user has access to the object

Access Control Lists

- Discretionary Access Control List (DACL):
  - ACE contain security ID and access masks
  - 2 types of ACEs in DACL:
    - Access allowed ACEs
    - Access denied ACEs



  - Accumulation of ACE‘s access form set of access rights granted by ACL
  - No DACL present -> everyone has full access
  - DACL is null (0 ACEs) -> no user has access to the object
- System Access Control List (SACL):
  - Contains only one type of ACE
  - Specifies which operations should be audited (stored in system audit log)

Access Control Entries (ACEs)

- Each ACE includes an access mask
  - Defines all possible actions for a particular object type
- Each object can have up to 16 *specific access types* (specific access mask)
- *Standard types* apply to all objects:
  - SYNCHRONIZE – allow a process to wait on signaled state,
  - WRITE_OWNER – assign write owner,
  - WRITE_DAC – write access to discretionary ACL,
  - READ_CONTROL – access to security descriptor,
  - DELETE – grant/deny delete access
- *Generic types*
  - FILE_GENERIC_READ, FILE_GENERIC_WRITE, FILE_GENERIC_EXECUTE

Assigning ACLs & Inheritance

1. Use security descriptor provided at object creation
2. Lookup security descriptor in object directory
   - For named objects only
   - Use security descriptors marked as inheritable to form ACL
3. If neither 1 or 2 apply:
   - Retrieve default ACL from caller‘s access token
   - Several subsystems have hard-coded DACLs that they assign on object creation (services, LSA, SAM objects)

Container objects can logically contain other objects

- New objects inside container object inherit permissions from parent
- Example: NTFS files inherit permissions from parent directory

Validate access to an object (1)

- Determine maximum access allowed to an object
  (NT 5.0 Win32 function *GetEffectiveRightsFromAcl()*)
  - Object has **no DACL** -> security system grants all access
  - Caller has **take-ownership privilege** -> security system grants write-owner access before examining DACL
  - **Caller is owner** -> read-control & write-control rights are granted
  - For each **access-denied ACE** that contains a SID that matches on in caller‘s access token, ACE‘s access mask is added to **denied-access** mask
  - For each **access-allowed ACE** that contains a SID that matches on in caller‘s access token, ACE‘s access mask is added to **granted-access** mask
    (unless that access has been denied)
- Granted access mask is returned as maximum allowed access to object

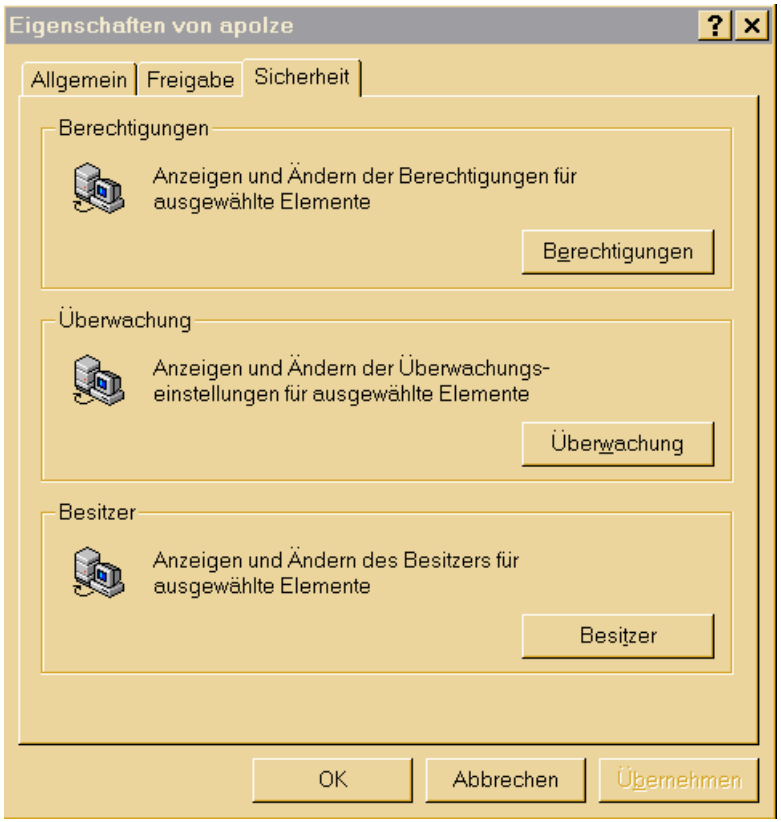
Validate access to an object (2)

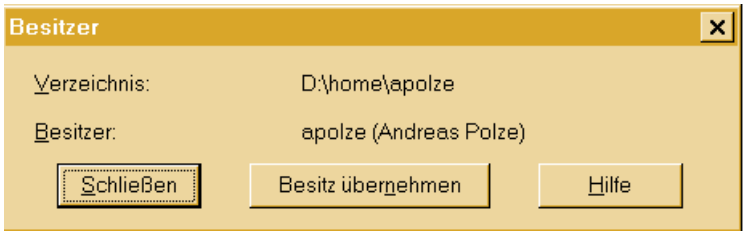
- Determine whether a specific access is allowed based on caller‘s access token and desired access mask (Win32 *AccessCheck(),* Windows 2000 *AccessCheckByType(), TrusteeAccessToObject()*)
  - Object has **no DACL** -> security system grants desired access
  - Caller has **take-ownership** -> write-owner access is granted before examining DACL (access is granted if it was the only access requested)
  - **Caller is owner** -> read-control & write-control DACL rights are granted (DACL is not examined if these were the only access rights requested)
  - Examine ACEs in ACL (see next page)
  - If end of DACL is reached end some access rights have not been granted, access is denied
- Access check is done when a handle is opened
  - No way to revoke access rights

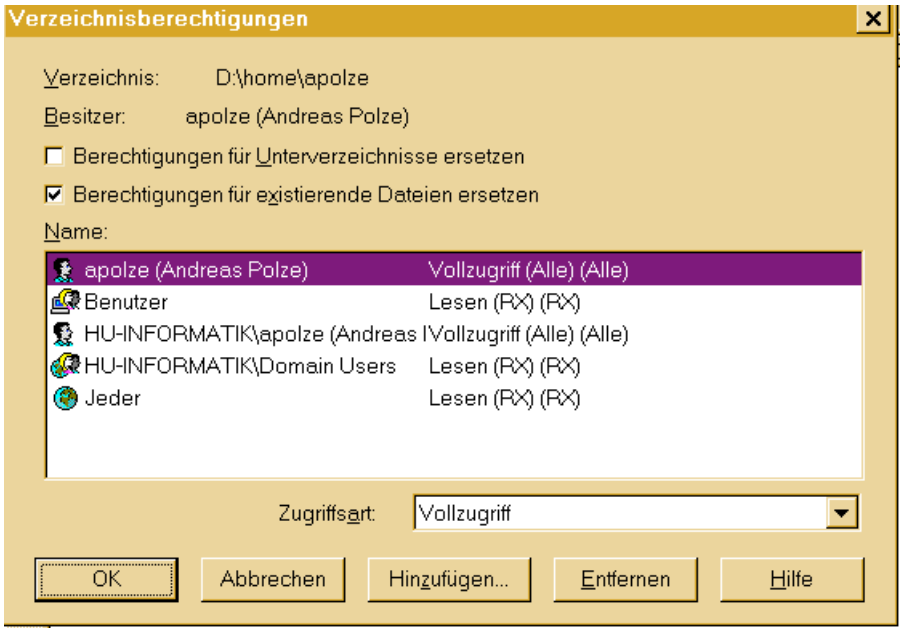
Validate access to an object (3)

- ACEs in DACL are examined, **first-to-last**,
  if SID in ACE matches enabled SID (primary or group SID) in callers access token:
  - Access-denied ACE: access to object is denied
  - Access-allowed ACE: granted rights (bits) are accumulated
    access check succeeds if all requested rights have been granted
- Convention:
  - Access-denied ACEs are placed before access-allowed ACEs
  - Win32 ACL functions allow to build ACL with ACE out of order
    - *Useful: emulate UNIX user/group/other-rights on NT files*
    - *See chown-Example*

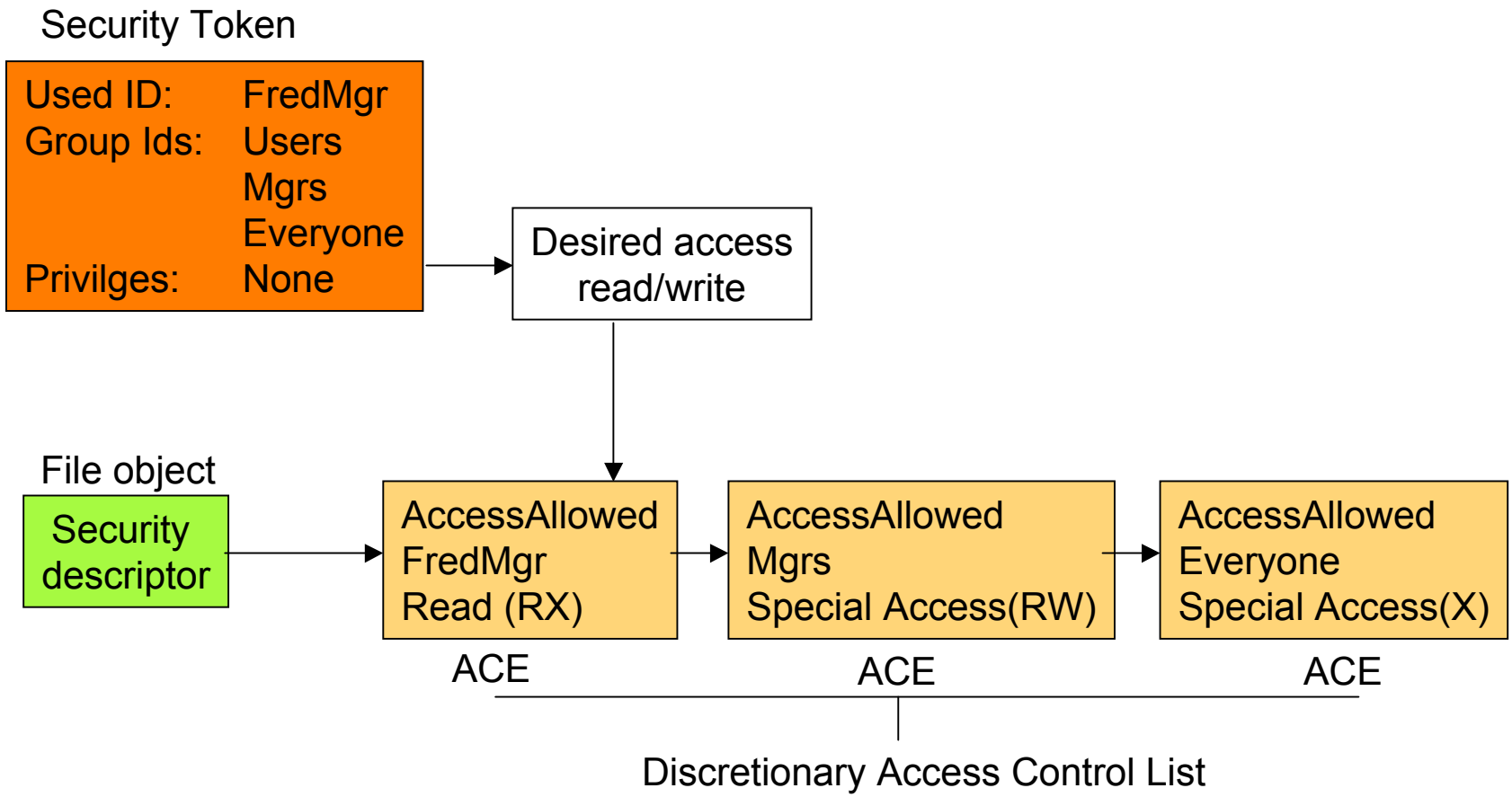
Example


Eigenschaften von apolze
Allgemein
Freigabe
Sicherheit
Berechtigungen
Anzeigen und Ändern der Berechtigungen für ausgewählte Elemente
Berechtigungen
Überwachung
Anzeigen und Ändern der Überwachungs-einstellungen für ausgewählte Elemente
Überwachung
Besitzer
Anzeigen und Ändern des Besitzers für ausgewählte Elemente
Besitzer
OK
Abbrechen
Übernehmen

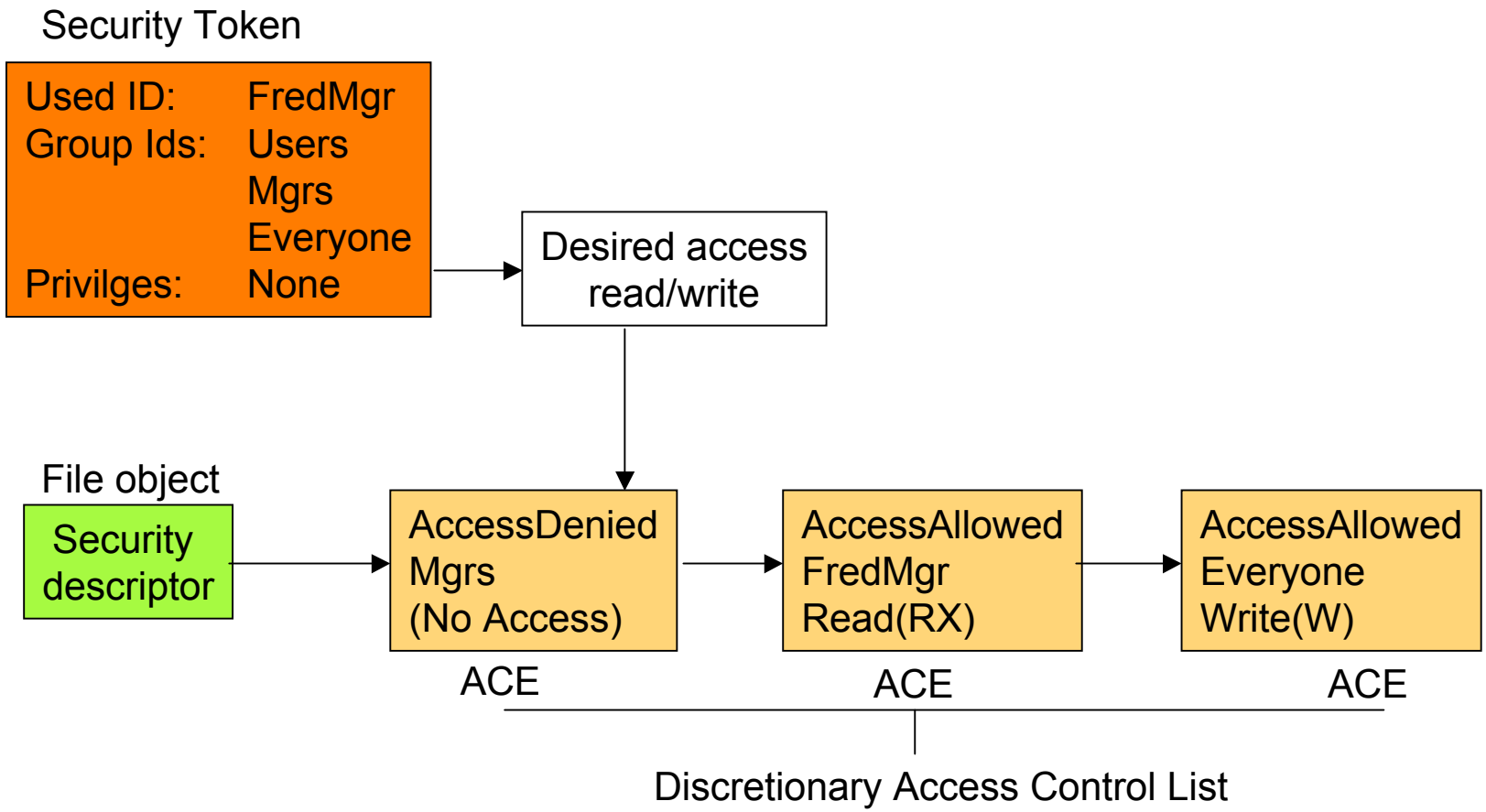

Besitzer
Verzeichnis: D:\home\apolze
Besitzer: apolze (Andreas Polze)
Schließen
Besitz übernehmen
Hilfe


Verzeichnisberechtigungen
Verzeichnis: D:\home\apolze
Besitzer: apolze (Andreas Polze)
Berechtigungen für Unterverzeichnisse ersetzen
Berechtigungen für existierende Dateien ersetzen
Name:
apolze (Andreas Polze) Vollzugriff (Alle) (Alle)
Benutzer Lesen (RX) (RX)
HU-INFORMATIK\apolze (Andreas I Vollzugriff (Alle) (Alle)
HU-INFORMATIK\Domain Users Lesen (RX) (RX)
Jeder Lesen (RX) (RX)
Zugriffsart: Vollzugriff
OK
Abbrechen
Hinzufügen...
Entfernen
Hilfe

Example: Access granted



Example: Access denied



Windows 2000: addl. Details

- Order of ACEs is more complicated:
  - Object-specific ACEs introduced
  - Automatic inheritance of ACEs
- Non-inherited ACEs go before inherited ACEs
- Within both groups, ACEs are placed according to their type:
  - Access-denied ACEs before access-allowed ACEs
  - Object-specific ACEs first, then sub-object specific ACEs, etc.

Access tokens

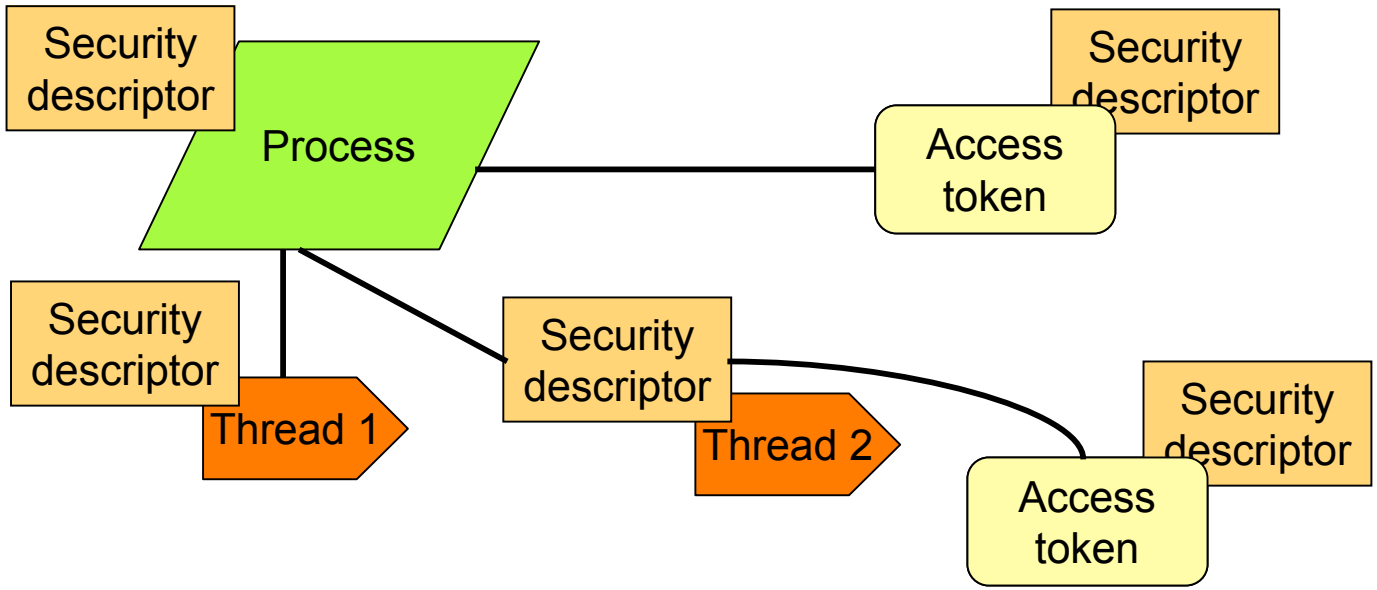
- Contains security identification of process or thread
  - Security ID (SID)
  - List of groups that the user is member of
  - List of privileges that are enabled/disabled
- Win32 functions create/manipulate access tokens
  - NT internal: object pointed to by process/thread block
- Processes inherit primary access token from creator
- At logon, LSASS verifies user/passwd and returns access token stored in SAM to WinLogon
  - WinLogon assigns the token to first user process
  - Win32 *LogonUser()* generates access token; can be used to create process with specific token via Win32 *CreateProcessAsUser()*

Impersonation

- Threads may have their own access token
  - If they represent a client (impersonate)
  - These threads may have different access token than process

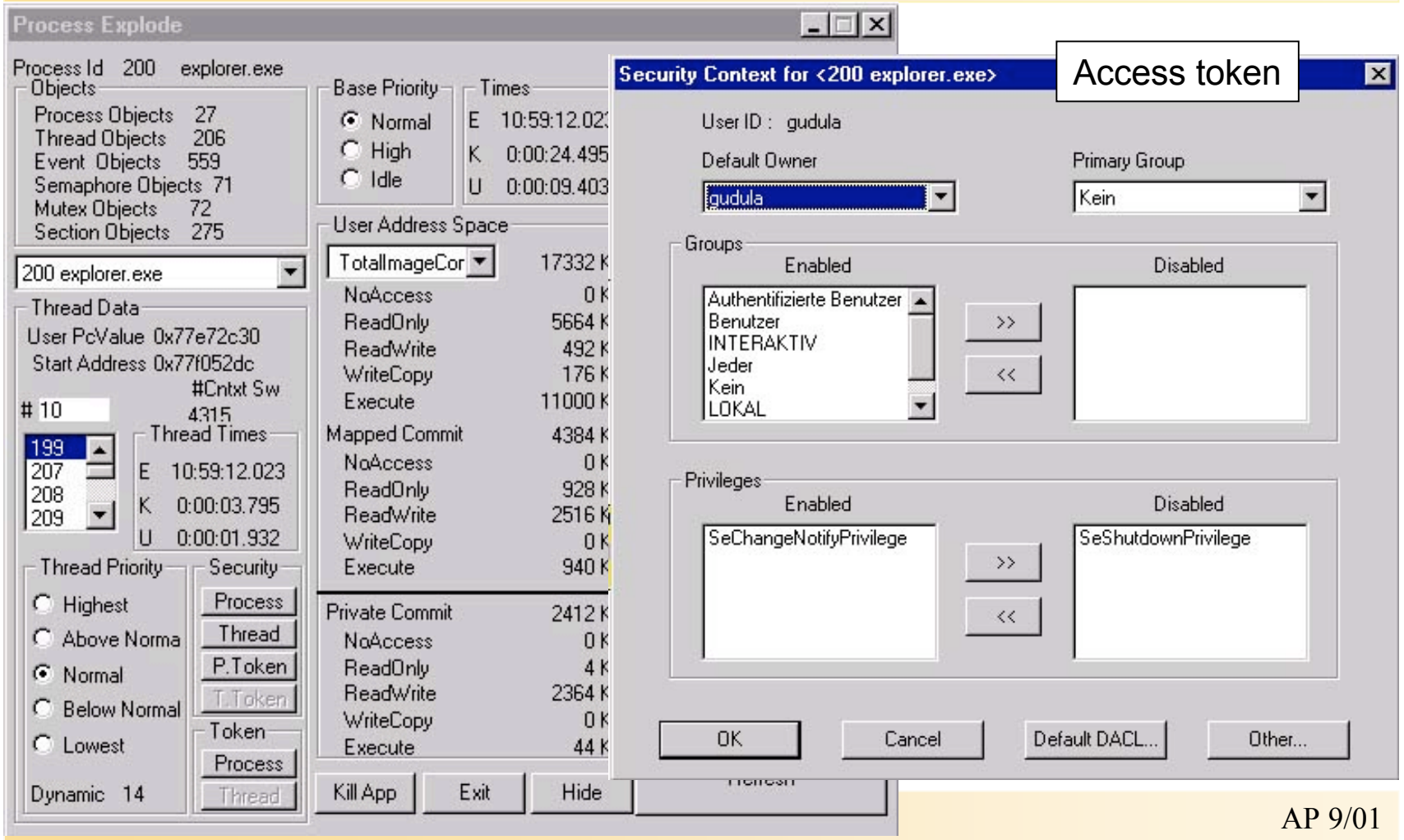
  -> Server threads may perform operations in client‘s security profile
  - Client can limit level of impersonation (security QoS)
    - SECURITY_ANONYMOUS, SECURITY_IDENTIFICATION, SECURITY_IMPERSONATION flags to *CreateFile()*
- Threads get own access token via
  - Win32 *ImpersonateSelf()* – clones process primary access token
  - Thread may take security token of client: *ImpersonateNamedPipeClient(), RpcImpersonateClient(), DdeImpersonateClient(), ImpersonateLoggedOnUser(), ImpersonateSecurityContext()* – see MSDN library

Process and Thread Security Structures



- Process/thread/access token objects have security descriptors
- Thread 2 has an impersonation token
- Thread 1 defaults to process access token

Experiment: Viewing Process and Thread Security Information



SYSTEM access token

- Many system processes (Services) run under special access token named SYSTEM
- Similar privileges as Administrators account in SAM

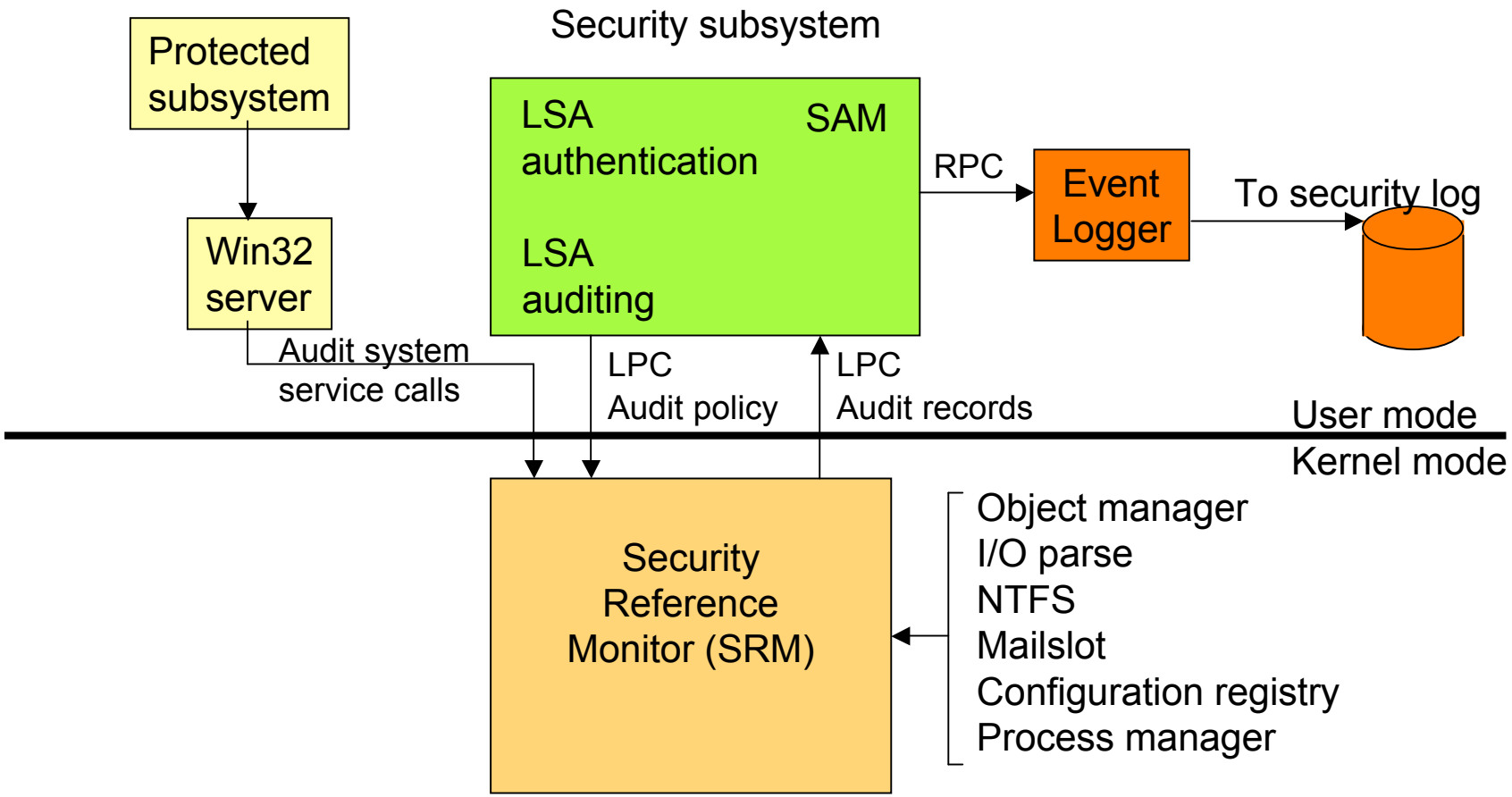
Restrictions for process under SYSTEM access token:

- No domain credential – no access to network resources
- Can‘t share objects with other non-SYSTEM user processes
  (unless it creates them using a NULL DACL or a DACL with explicit access rights for user/group)

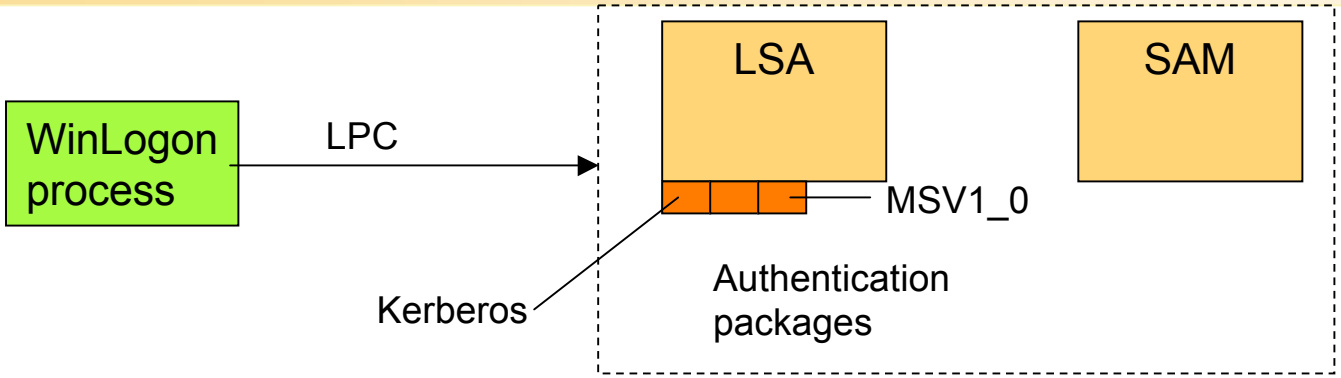
Security Auditing

- Object manager can generate audit events as result of an access check
- Applications can generate events via Win32 func.
- Processes that call audit system services must have *SeAuditPrivilege* – prevent flooding security log
- Audit policy of local system decides which events to log
  - Maintained by LSA
  - LSA sends messages to SRM to inform about auditing policy (at system startup and when policy changes)
  - LSA receives audit messages from SRM and sends it to Event Log
  - LSA and SAM generate own audit events
  - LPC or shared memory communication between LPC SRM, LSA, SAM, and Event Logger (depending on message size)

Flow of security audit records



Logon



- WinLogon is trusted process
  - Intercepts logon requests from keyboard
  - Calls LSA
- Identification/authentication aspects in replacable DLL:
  - MSGINA.DLL (default Graphical Id. & Auth.)
  - Developers can bring in their own GINA (logon via SmartCards, etc.)

WinLogon Initialization

At system initialization, WinLogon performs these steps:

- Create & open window station:
  - Represent keyboard, mouse, monitor
  - Create SID with only one ACE containing WinLogon SID
  - No process can access workstation unless allowed by WinLogon
- Create & open three desktops:
  - Application, WinLogon, screen saver desktops
  - Only WinLogon can access logon desktop (SID)
  - No other process has access to code/data connected to logon desktop
- Establish LPC connection with LSA
  - Via *LsaRegisterLogonProcess()*
  - Call *LsaLookupAuthenticationPackage()* to get association ID for MSV1_0/Kerberos to be used for logon authentication

WinLogon: Set up Window Environment

- Initialize/register window class data structure
  - Associate WinLogon procedure with its windows
- Register secure attention sequence (SAS – ctrl-alt-del)
  - Associate SAS with WinLogon window
  - WinLogon gets control of screen whenever SAS is entered (avoid Trojan horses)
- Register the window for log off/screen saver timeout
  - Win32 subsystem checks to verify that process requesting notification is the WinLogon process
- After initialization, WinLogon desktop is active
  - Locked by WinLogon, unlocked only to switch to application/screen screen desktops

User Logon Steps

- User presses SAS
  - WinLogon switches to secure desktop
  - Prompts for username/password
  - Creates a unique local group for this user (for keyboard, screen, mouse)
  - WinLogon passes this group to LSA (*LsaLogonUser()*)
  - This group will be included in logon process token
- LSA calls authentication package with user/passwd
  - MSV1_0 implements Windows 2000 authentication (stand-alone syst.)
  - Kerberos implements auth. for members of Windows 2000 domain
  - All packages on the system are in the registry at HKLM\System\CurrentControlSet\Control\Lsa
- MSV1_0 takes user/passwd and sends request to SAM
  - SAM retrieves account info: passwd, groups, account restrictions

User Logon Steps (contd.)

- MSV1_0 checks account restrictions
  - Hours or type of access allowed
  - MSV1_0 compares passwd and generates unique ID for logon session (logon user ID – LUID)
  - Creates logon session by passing LUID + addl. info. to LSA
- LSA looks in local policy database
  - Logon will be terminated if user‘s requested access is not allowed (interactive, network, service process)
- LSA accumulates info for access token
  - Includes user‘s SID, group SIDs, user profile info (home dir...)
  - Includes addl. security Ids/privileges (Everyone, Interactive, etc.)
- Executive creates access token; passes it to LSA
  - Primary token: interact./service logon; impersonation token: netw. logon
  - LSA duplicates token, passes handle to WinLogon (+LUID, profile info)