Unit OS7: Security

7.3. Windows Security Descriptors

Roadmap for Section 7.3.

- Protecting Objects
- Security Descriptors and Access Control Lists
- Auditing and Impersonation
- Privileges

Protecting Objects

- Access to an object is gated by the Security Reference Monitor (SRM),
  - performs access validation at the time that an object is opened by a process
- Access validation is a security equation that consists of the following components:
  - Desired Access: the type of access that is being requested.
    - must be specified up front,
    - include all accesses that will be performed on the object as a result of the validation.
  - Token: identifies the user that owns the process, as well as the privileges of the user.
    - Threads can adopt a special type of token called an "impersonation token" that contains the identify of another account.
  - The object's Security Descriptor
    - contains a Discretionary Access Control List (DACL),
    - describes the types of access to the object users are allowed.

4

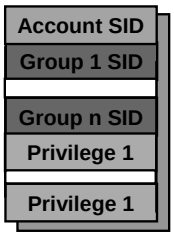
Handles and Security

- If the validation succeeds, a handle is created in the process requesting access and through which the process accesses the resource
- Changing security on an object only affects subsequent opens
  - Processes that have existing handles can continue to access objects with the accesses they were granted
  - E.g. changing permissions on a share won't affect currently connected users
- Lab: View process handles and corresponding granted accesses with Process Explorer

5

Tokens

- The main components of a token are:
  - SID of the user
  - SIDs of groups the user account belongs to
  - Privileges assigned to the user (described in next section)



6

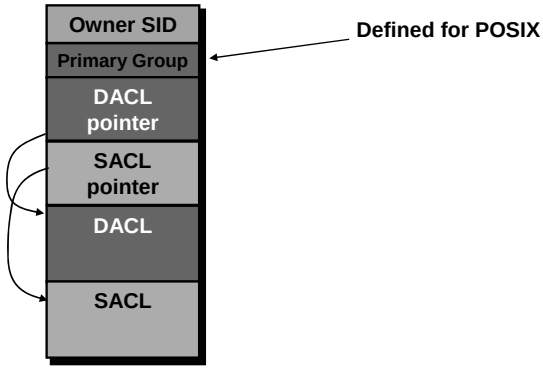
Security Identifiers - SIDs

- Windows uses Security Identifers (SIDs) to identify security principles:
  - Users, Groups of users, Computers, Domains
- SIDs consist of:
  - A revision level e.g. 1
  - An identifier-authority value e.g. 5 (SECURITY_NT_AUTHORITY)
  - One or more subauthority values
- SIDs are generally long enough to be globally statistically unique
- Setup assigns a computer a SID
- Users and groups on the local machine are assigned SIDs that are rooted with the computer SID, with a Relative Identifier (RID) at the end
  - Some local users and groups have pre-defined SIDs (eg. World = S-1-1-0)
  - RIDs start at 1000 (built-in account RIDs are pre-defined)

7

Security Descriptors

- Descriptors are associated with objects: e.g. files, Registry keys, application-defined
- Descriptors are variable length



8

Win32 Security

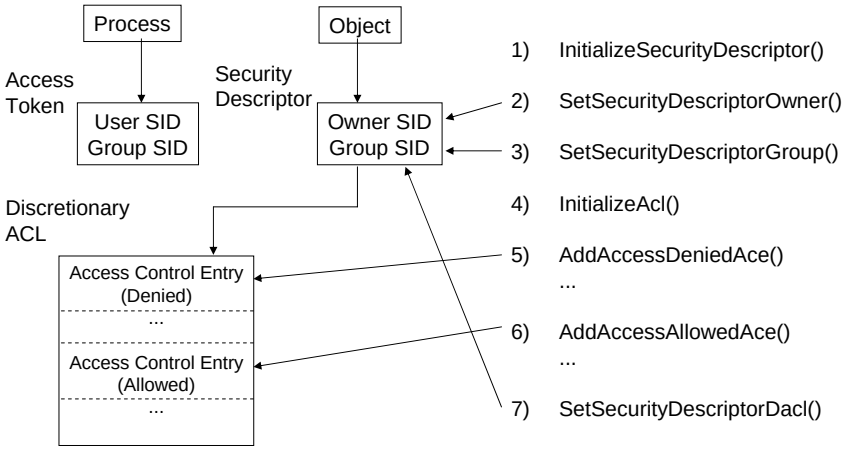
- Comprehensive security model
- Nearly all shareable objects can be protected
- Programmer has fine granularity of control over access rights
- Security attributed may be specified at object creation

```
typedef struct _SECURITY_ATTRIBUTES {
        DWORD nLength;
        LPVOID lpSecurityDescriptor;
        BOOL bInheritHandle;
} SECURITY_ATTRIBUTES;
```

```
nLength = sizeof(
 SECURITY_ATTRIBUTES);
bInheritHandle = FALSE;
```

9

Constructing a Security Descriptor



10

Access Control Lists

- ACL is a set of Access Control List Entries (ACEs)
  - 2 types: access-allowed / access-denied
- Initialize ACL with InitializeAcl()

Each ACE contains SID and *access mask*

- Add ACEs to DACL with
  - AddAccessAllowedAce()
  - AddAccessDeniedAce()
- Add ACEs to SACL with
  - AddAuditAccessAce()
- Manage ACEs with
  - DeleteAce() and GetAce()

Order of ACEs in ACL is important:
First-Fit alg.
- frequently: access-denied ACEs first
- other schemes possible

11

Security Identifiers (SIDs)

- Win32 uses SIDs to identify users and groups
- SID can be obtained from account name
- Account can be on remote system:

```
BOOL LookupAccountName( LPCTSTR lpszSystem,
        LPCTSTR lpszAccount,
        PSID psid,
        LPDWORD lpcbSid,
        LPTSTR lpszReferencedDomain,
        LPDWORD lpcchReferencedDomain,
        PSID_NAME_USE psnu );
```

- If SID is given account name can be obtained using LookupAccountSid()

12

Example

- UNIX-Style Permissions for NTFS Files
- chmod / lsFP commands
  - InitializeUnixSA creates valid security attributes structure
  - ReadFilePermissions
  - ChangeFilePermissions
- To emulate UNIX behavior, order of access-allowed / access-denied ACEs is critical
- Ugo-bits = 466 – user has read but no write access rights; even if user is member of group

13

Read Access Control Entries

```
/* Get the required size for the security descriptor. */
GetFileSecurity (lpFileName, OWNER_SECURITY_INFORMATION |
        GROUP_SECURITY_INFORMATION | DACL_SECURITY_INFORMATION,
        pSD, 0, &LenNeeded);

/* Create a security descriptor. */
pSD = HeapAlloc (ProcHeap, HEAP_GENERATE_EXCEPTIONS, LenNeeded);
if (!GetFileSecurity (lpFileName, OWNER_SECURITY_INFORMATION |
        GROUP_SECURITY_INFORMATION | DACL_SECURITY_INFORMATION,
        pSD, LenNeeded, &LenNeeded))
        ReportError (_T ("GetFileSecurity error"), 30, TRUE);

if (!GetSecurityDescriptorDacl (pSD, &DaclF, &pAcl, &AclDefF))
        ReportError (_T ("GetSecurityDescriptorDacl error"), 31, TRUE);
```

14

Read Access Control Entries (contd.)

```
/* Get the number of ACEs in the ACL. */
if (!GetAclInformation (pAcl, &ASizeInfo, sizeof (ACL_SIZE_INFORMATION),
        AclSizeInformation))
        ReportError (_T ("GetAclInformation error"), 32, TRUE);

/* Get Each Ace. Accumulate permission bits. */
PBits = 0;
for (iAce = 0; iAce < ASizeInfo.AceCount; iAce++) {
        GetAce (pAcl, iAce, &pAce);
        AType = pAce->Header.AceType;
        if (AType == ACCESS_ALLOWED_ACE_TYPE)
                PBits |= (0x1 << (8-iAce));
}
```

15

Obtain Security IDs

```
/* Find the name of the owner and owning group. */
/* Find the SIDs first. */

if (!GetSecurityDescriptorOwner (pSD, &pOwnerSid, &OwnerDefF))
        ReportException (_T ("GetSecurityDescOwner error"), 33);
if (!GetSecurityDescriptorGroup (pSD, &pGroupSid, &GroupDefF))
        ReportException (_T ("GetSecurityDescGrp error"), 34);
if (!LookupAccountSid (NULL, pOwnerSid, UsrNm, &AcctSize [0],
        RefDomain [0], &RefDomCnt [0], &sNamUse [0]))
        ReportException (_T ("LookUpAccountSid error"), 35);
if (!LookupAccountSid (NULL, pGroupSid, GrpNm, &AcctSize [1],
         RefDomain [1], &RefDomCnt [1], &sNamUse [1]))
        ReportException (_T ("LookUpAccountSid error"), 36);
```

16

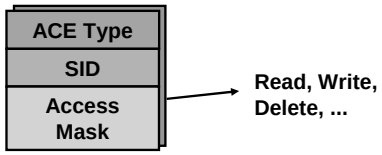
Access Control Entries (ACEs)

- 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

17

Discretionary Access Control Lists DACLs

- DACLs consist of zero or more Access Control Entries
  - A security descriptor with no DACL allows all access
  - A security descriptor with an empty (0-entry) DACL denies everybody all access
- An ACE is either “allow” or “deny”



18

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

19

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

20

Validate access to an object (2)

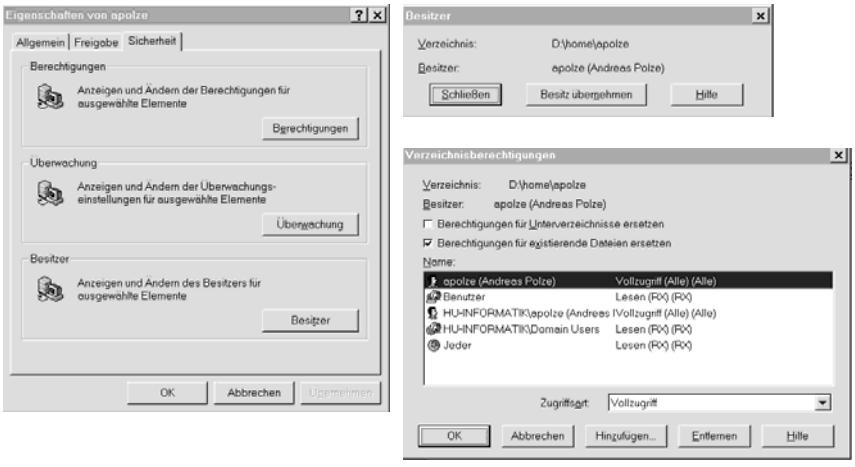
- Determine whether a specific access is allowed based on caller‘s access token and desired access mask
  (Win32 *AccessCheck(),* Windows 2000 *AccessCheckByType(), TrusteeAccessToObject()*)
  - Object has **no DACL** -> security system grants desired access
  - Caller has **take-ownership** -> write-owner 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

21

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

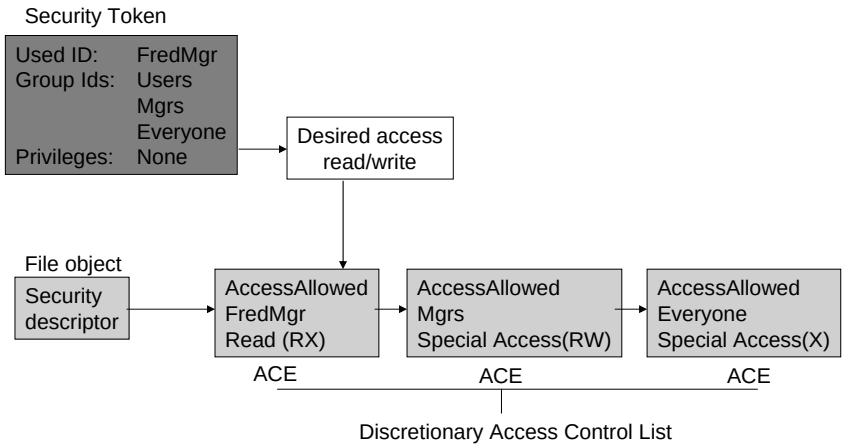


Access Check - recap

- ACEs in the DACL are examined in order
  - Does the ACE have a SID matching a SID in the token?
  - If so, do any of the access bits match any remaining desired accesses?
  - If so, what type of ACE is it?
    - Deny: return ACCESS_DENIED
    - Allow: grant the specified accesses and if there are no remaining accesses to grant, return ACCESS_ALLOWED
  - If we get to the end of the DACL and there are remaining desired accesses, return ACCESS_DENIED
- The Security Reference Monitor (SRM) implements an *explicit allow* model
  - Exposed to apps through Windows API AccessCheck(), AccessCheckByType(), TrusteeAccessToObject())

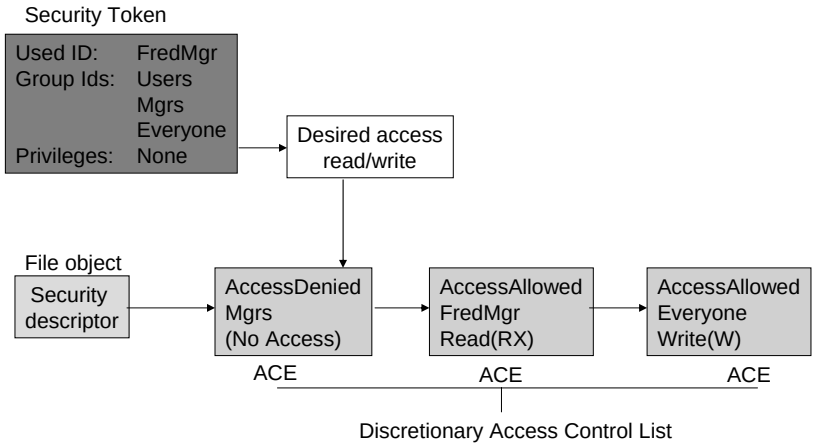
24

Example: Access granted



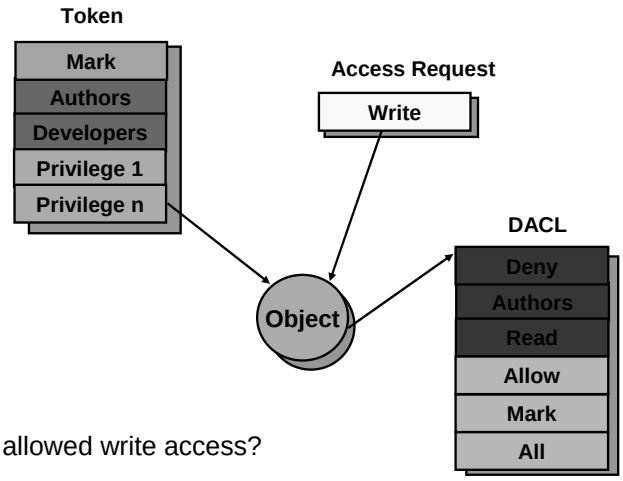
25

Example: Access denied



26

Access Check Quiz



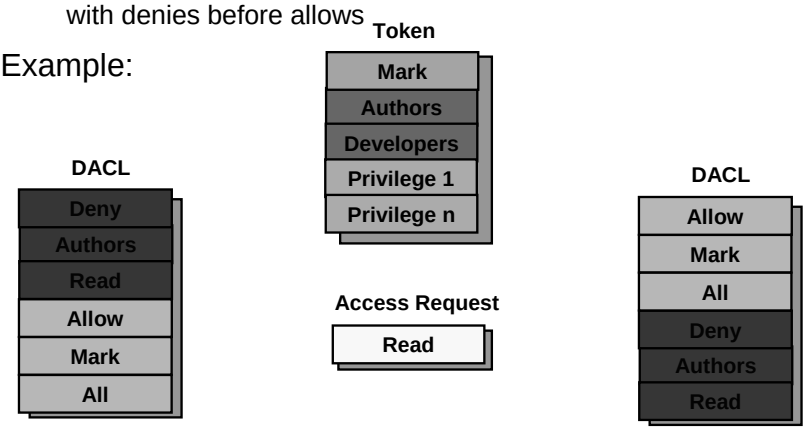
Is Mark allowed write access?

yes

27

ACE Ordering

- The order of ACEs is important!
  - Low-level security APIs allow the creation of DACLs with ACEs in any order
  - All security editor interfaces and higher-level APIs order ACEs with denies before allows
- Example:



28

Access Special Cases

- An object's owner can always open an object with WRITE_DACL and READ_DACL permission
- An account with "take ownership" privilege can claim ownership of any object
- An account with backup privilege can open any file for reading
- An account with restore privilege can open any file for write access

29

Object-specific ACEs

- Object-specific ACEs can be applied to Directory Services (DS) objects
  - They are just like ACES, but have two GUID fields
- The GUIDs allow the ACE to:
  - Control access to a property sheet or set on the object
  - Specify the type of child object that can inherit the ACE
  - Specify the type of child object for which the ACE grants or denies creation rights

30

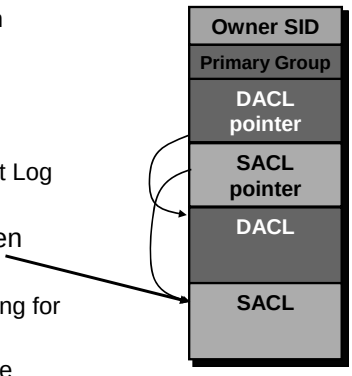
Controllable Inheritance

- In NT 4.0, objects only inherit ACEs from a parent container (e.g. Registry key or directory) when they are created
  - No distinction made between inherited and non-inherited ACES
  - No prevention of inheritance
- In Windows 2000 and higher inheritance is controllable
  - SetNamedSecurityInfoEx and SetSecurityInfoEx
  - Will apply new inheritable ACEs to all child objects (subkeys, files)
  - Directly applied ACEs take precedence over inherited ACEs

31

Auditing

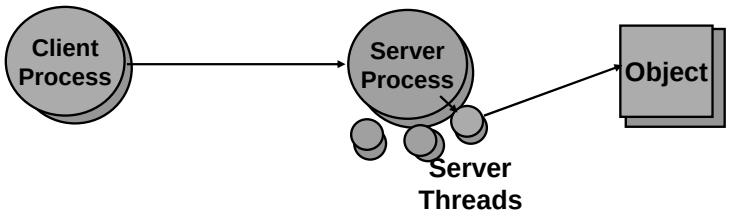
- Provides for monitoring of accesses to objects
  - Even if you specify auditing information for an object, it won't result in audit records unless Auditing is enabled
  - An administrator can enable it with the Local Security Policy Editor (secpol.msc)
  - The security log can be viewed with the Event Log Viewer
- Like for DACLs, SACL check is made on open after access check
  - Audit check is performed only if system auditing for access check result is on
  - Only ACEs that match access check result are processed
  - Test is similar to DACL test, but a record is written if there is any match
- Demo: Explorer file auditing settings



32

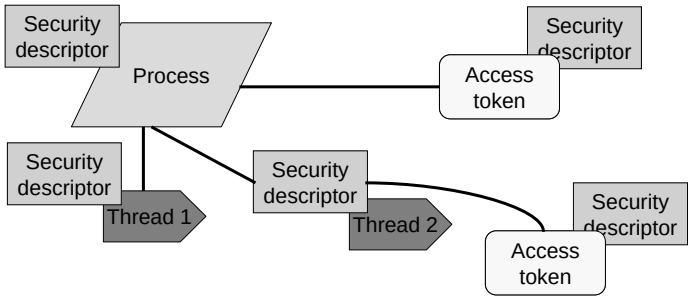
Impersonation

- Lets an application adopt the security profile another user
  - Used by server applications
  - Impersonation is implemented at the thread level
    - The process token is the "primary token" and is always accessible
    - Each thread can be impersonating a different client
- Can impersonate with a number of client/server networking APIs – named pipes, RPC, DCOM



33

Process and Thread Security Structures

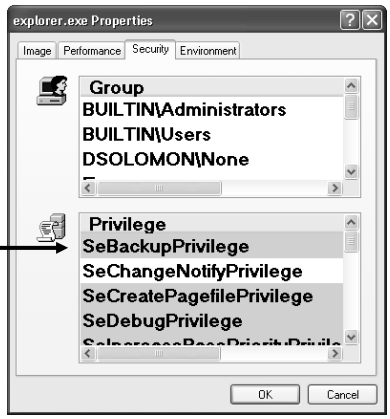


- Process/thread/access token objects have security descriptors
- Thread 2 has an impersonation token
- Thread 1 defaults to process access token

34

Privileges

- Specify which system actions a process (or thread) can perform
- Privileges are associated with groups and user accounts
  - There are sets of pre-defined privileges associated with built-in groups (e.g. System, Administrators)
- Examples include:
  - Backup/Restore
  - Shutdown
  - Debug
  - Take ownership
- Privileges are disabled by default and must be programmatically turned on with a system call



35

Powerful Privileges

- There are several privileges that gives an account that has them full control of a computer:
  - Debug: can open any process, including System processes to
    - Inject code
    - Modify code
    - Read sensitive data
  - Take Ownership: can access any object on the system
    - Replace system files
    - Change security
  - Restore: can replace any file
  - Load Driver
    - Drivers bypass all security
  - Create Token
    - Can spoof any user (locally)
    - Requires use of undocumented Windows API
  - Trusted Computer Base (Act as Part of Operating System)
    - Can create a new logon session with arbitrary SIDs in the token

36

Further Reading

- Mark E. Russinovich and David A. Solomon, Microsoft Windows Internals, 4th Edition, Microsoft Press, 2004.
  - Chapter 8, Security
  - Security Descriptors and Access Control (from pp. 506)
  - Account Rights and Privileges (from pp. 516)
- Johnson M. Hart, Win32 System Programming: A Windows 2000 Application Developer's Guide, 2nd Edition, Addison-Wesley, 2000.
  - Chapter 5, Securing Win32 objects (from pp. 111)

37

Source Code References

- Windows Research Kernel sources
  - \base\ntos\se – Security Reference Monitor
    - Accessck.c – central access check function
    - Token*.*, Seclient.c – Token support
    - Seaudit.c, Adt*.* – security auditing
    - Privileg.c, Seastate.c – User right/privilege management
    - Sep.h – private structure/type definitions
  - \base\ntos\inc\se.h – additional structure/type definitions

38