Unit 6: Protection and Security

6.5. Win32 Security Descriptors

Win32 Security

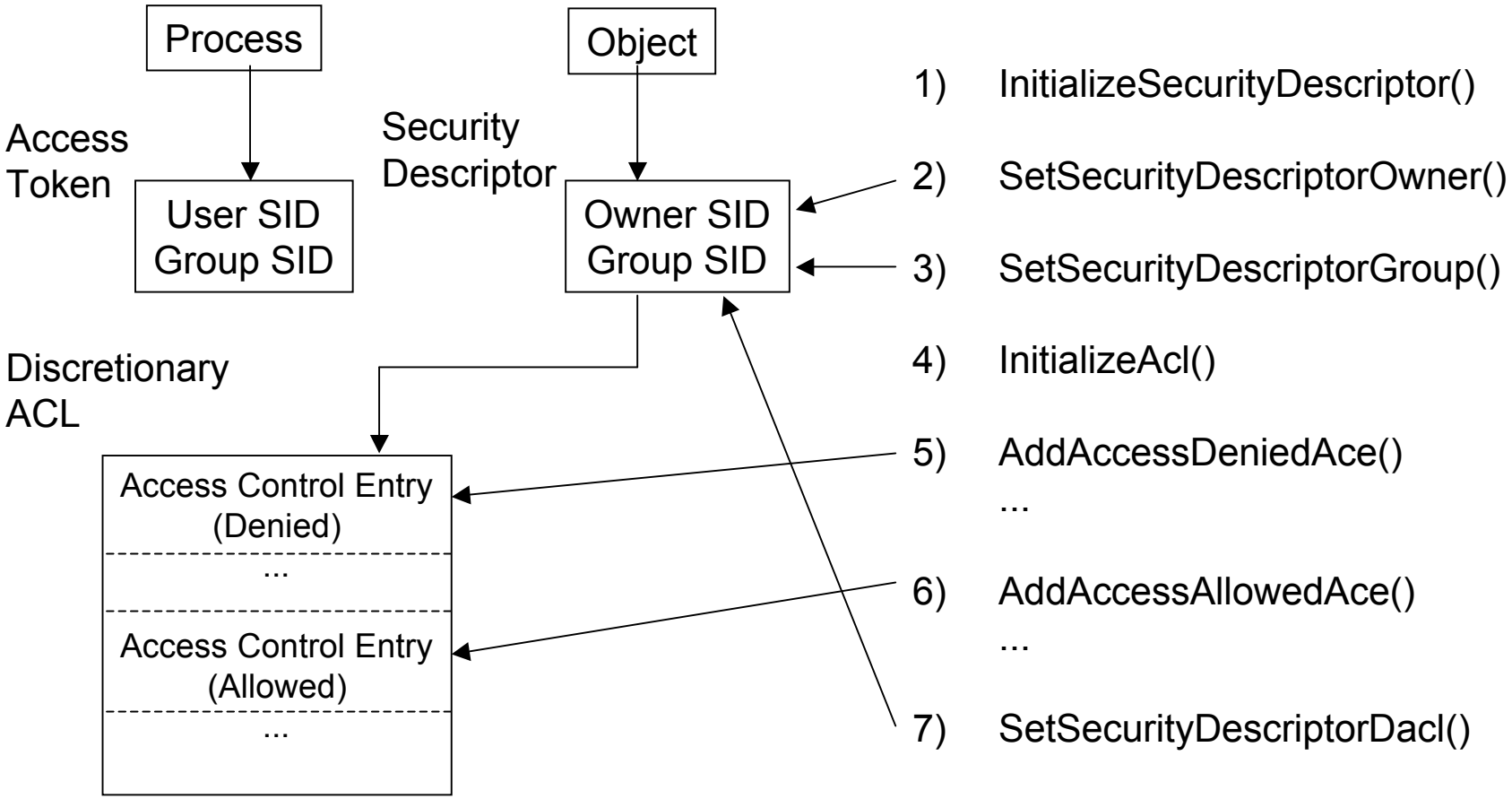
- Comprehensive security model
- Nearly all shareable objects can be protected
- Programmer has fine granularity of control over access rights

- 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;
```

Constructing a Security Descriptor



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

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()

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

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);
```

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));
}
```

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);
```