

[MS-SECO]: Windows Security Overview

There are many concepts ingrained in the Windows Security Overview that result from years of pervasive use. For someone new to Windows, this can be a source of frustration, because references can exist in many places and may not be presented in a simple manner. This document is intended to explain fundamental concepts and to point readers to more information.

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft [Open Specification Promise](#) or the [Community Promise](#). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard

specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
02/14/2008	2.0.3	Editorial	Revised and edited the technical content.
03/14/2008	2.0.4	Editorial	Revised and edited the technical content.
06/20/2008	2.0.5	Editorial	Revised and edited the technical content.
07/25/2008	3.0	Major	Updated and revised the technical content.
08/29/2008	3.0.1	Editorial	Revised and edited the technical content.
10/24/2008	3.1	Minor	Updated the technical content.
12/05/2008	3.2	Minor	Updated the technical content.
01/16/2009	3.2.1	Editorial	Revised and edited the technical content.
02/27/2009	3.2.2	Editorial	Revised and edited the technical content.
04/10/2009	3.2.3	Editorial	Revised and edited the technical content.
05/22/2009	3.2.4	Editorial	Revised and edited the technical content.
07/02/2009	3.2.5	Editorial	Revised and edited the technical content.
08/14/2009	4.0	Major	Updated and revised the technical content.
09/25/2009	4.0.1	Editorial	Revised and edited the technical content.
11/06/2009	4.1	Minor	Updated the technical content.
12/18/2009	4.1.1	Editorial	Revised and edited the technical content.
01/29/2010	4.1.2	Editorial	Revised and edited the technical content.
03/12/2010	5.0	Major	Updated and revised the technical content.
04/23/2010	5.0.1	Editorial	Revised and edited the technical content.
06/04/2010	5.0.2	Editorial	Revised and edited the technical content.
07/16/2010	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
08/27/2010	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
10/08/2010	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.

Date	Revision History	Revision Class	Comments
11/19/2010	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
01/07/2011	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
02/11/2011	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
03/25/2011	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
05/06/2011	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.
06/17/2011	5.0.2	No change	No changes to the meaning, language, or formatting of the technical content.

Contents

1	Introduction	6
2	Identity	7
2.1	Security Principal	7
2.2	Accounts	7
2.2.1	User accounts	8
2.2.2	Computer accounts	8
2.3	Security Identifiers (SIDs)	9
2.4	Groups	11
2.4.1	Group Types	12
2.4.2	Group Scope	12
2.4.3	Nested Groups	13
2.5	Account Domains	14
2.5.1	Local Domains	14
2.5.2	Remote Domains and Domain Controllers	14
2.5.3	Domain Membership	15
2.5.4	Effect on Accounts	15
2.5.5	Globally Unique Identifiers (GUIDs)	16
2.5.5.1	Uniqueness	16
2.5.5.2	Internal Structure	16
2.5.5.3	Quality of Random Bits	17
3	Authentication	19
3.1	Background	19
3.1.1	Authentication Concepts	19
3.1.2	Practice	19
3.1.3	History	20
3.2	Authentication Protocols	20
3.2.1	NT LAN Manager (NTLM)	20
3.2.1.1	Description and Flow	20
3.2.1.2	Notes	22
3.2.2	Kerberos	23
3.2.2.1	Description and Flow	23
3.2.2.2	Notes	24
3.2.3	Secure Sockets Layer (SSL) and Transport Layer Security (TLS)	25
3.2.4	Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)	25
3.2.4.1	Description and Flow	25
3.2.4.2	Notes	26
3.3	Use of Authentication	26
3.3.1	General Model	26
3.3.2	Known Idiosyncrasies	26
3.3.2.1	Server Message Block (SMB) and Common Internet File System (CIFS)	27
4	Authorization	28
4.1	Resource Managers	28
4.2	Security Descriptors	28
4.3	Discretionary Access Control Lists (DACLS) and Access Control Entries (ACEs)	29
4.4	Access Rights	31
4.5	Authorization	31
4.6	Inheritance	32

4.7	System Access Control Lists	32
4.8	Windows Integrity Mechanism	33
5	Impersonation	34
5.1	Cloaking	35
5.2	Impersonation Tokens	36
5.3	Impersonation Levels	36
5.4	Setting the Impersonation Level	37
5.5	Windows API Impersonation Functions	38
6	References	39
7	Appendix A: Product Behavior	40
8	Change Tracking.....	42
9	Index	43

1 Introduction

The Microsoft Windows® operating system contains a wide range of security concepts, which can be overwhelming when first encountered. Windows is a relatively mature product with regard to security concepts; some technologies extend back 15 years or more. Microsoft is also seeking to adopt new security technologies and concepts and to incorporate them into a model that is already familiar to many system administrators.

2 Identity

The first security concept that is important to understand is identity. Identity in Microsoft Windows® comes in several types, and it exists and is managed in several scopes, which are detailed in the following sections. For example, identity can refer to the set of users on a single computer or the identities that are available in a domain.

2.1 Security Principal

A security principal is a common concept in security; it is an actor in a security system and is often something capable of initiating action. Typically, a security principal is associated with a human user of the computer system, but it can also be an autonomous program within the system, such as a logging daemon, system backup program, or something similar.

The security principal is an entity that can be authenticated. In Microsoft Windows®, a security principal is typically a user but can also be a computer or a service. A security principal is often referred to as an *account*.

Security principals receive permissions to access resources such as files and folders. User rights, such as interactive logons, are granted or denied to accounts directly or via membership in a group. The accumulation of these permissions and rights defines what security principals can and cannot do when working on the network.

2.2 Accounts

One of the most important aspects of any security principal is that it serves as a point of management between the system and the administrator. As such, it needs to have attributes that make it meaningful to the administrator or the user. The security principal (or account) has at least a *name* and an *identifier*.

The name is a simple textual name for the account, such as John Smith, SYSTEM, or RedmondDc1\$. The name is merely an attribute of the account, however, and can change over time. A common scenario is that the person that the account refers to changes his or her name.

Also, the name is treated as case-*insensitive*. That is, John Smith, JOHN SMITH, john smith, and joHn SmiTH are treated as equivalent in Microsoft Windows®. Microsoft views case-sensitivity as creating an unnecessary burden on the administrator and as something that can lead to mistakes.

The identifier, though also an attribute of the account, has to satisfy other attributes as well. Of particular importance are the uniqueness and persistence of the identifier and the issuer of the identifier. The persistence of the identifier is what provides the administrator with the capability to assign a resource to that account and not be surprised in the future by changes to the account.

Consider the case of John Smith. The administrator may assign John Smith access to a certain document at one point in time. If that John Smith leaves the company and a new John Smith is hired, the new John Smith should not have access to the resources of the original John Smith. Conversely, if John Smith changes his name to John Q. Smith, he should not lose access to the resources previously granted.

The other important attribute is the issuer of the identifier. Identities have different weight, conceptually, depending on the issuer. In the physical world, a store is generally willing to accept a driver's license as proof of identity, but the store is unwilling to accept a gymnasium membership card. In the Windows model, the issuer of an account is encoded with the identity so that any recipient can make a similar decision.

The identifier that Windows uses for accounts is called a *security identifier (SID)*.

Windows contains a number of built-in accounts:

- **User account:** Identifies users who belong to the domain by storing their names, their passwords, the groups that they belong to, the permissions that they have for accessing system resources, and other personal information.
- **Group account:** Identifies a specific group of users and is used to assign them permissions to objects and resources.
- **Computer account:** Identifies computers that belong to the domain. A computer account is commonly referred to as a "machine account."

Each built-in user, computer, or group account is a security principal.[<1>](#)

User and computer accounts can be added, disabled, reset, and deleted by using Active Directory Users and Computers. A computer account can also be created when a computer is joined to a domain. For more information about user and computer accounts, see Active Directory naming and Object names.

2.2.1 User accounts

In the Active Directory directory service, each user account has a user logon name, a pre-Microsoft Windows® 2000 operating system user logon name (security accounts manager (SAM) account name), and a user principal name (UPN) suffix. The administrator enters the user logon name and selects the UPN suffix when creating the user account. Active Directory suggests a pre-Windows 2000 user logon name that uses the first 20 bytes of the user logon name. Administrators can change the pre-Windows 2000 logon name at any time.

In Active Directory, each user account has a UPN based on [\[RFC822\]](#). The UPN is composed of the user logon name and the UPN suffix joined by the at sign (@).

Note When creating a user account, it is not necessary to add the at sign (@) to the user logon name or to the UPN suffix. Active Directory automatically adds the at sign (@) when it creates the UPN. A UPN that contains more than one at sign (@) is invalid. User logon names do not end with a period or multiple periods.[<2>](#)

The second part of the UPN, the UPN suffix, identifies the domain in which the user account is located. This UPN suffix can be the Domain Name System (DNS) domain name, the DNS name of any domain in the forest, or an alternative name that is created by an administrator and used just for logon purposes. This alternative UPN suffix does not need to be a valid DNS name.

In Active Directory, the default UPN suffix is the DNS name of the domain in which the user account is created. In most cases, this is the domain name that is registered as the enterprise domain on the Internet. Using alternative domain names as the UPN suffix can provide additional logon security and simplify the names that are used to log on to another domain in the forest.

For example, if an organization uses a deep domain tree that is organized by department and region, domain names can become quite long. The default UPN for a user in that domain might be sales.westcoast.contoso.com. The logon name for a user in that domain would be user@sales.westcoast.contoso.com. Creating a UPN suffix of "contoso" would allow that same user to log on by using the much simpler logon name of user@contoso.

2.2.2 Computer accounts

Each computer account that is created in Active Directory has a relative distinguished name (RDN), a pre-Microsoft Windows® 2000 operating system computer name (SAM account name), a primary

DNS suffix, a DNS host name, and a service principal name (SPN) in addition to the computer name. The administrator enters the computer name when creating the computer account. The computer name must include the dollar sign (\$) character at the end of the name (for example RedmondDc1\$).

When the domain functional level has been set to Windows Server® 2003 operating system, Windows Server® 2003 R2 operating system, Windows Server® 2008 operating system, and Windows Server® 2008 R2 operating system, a new **lastLogonTimestamp** attribute is used to track the last logon time of a user or computer account. This attribute is replicated in the domain and can provide important information regarding the history of a user or a computer.

Every Microsoft Windows® computer that joins a domain has a computer account. [<3>](#) Similar to user accounts, computer accounts provide a means for authenticating and auditing computer access to the network and to domain resources. Each computer account must be unique.

When the Netlogon service running on a client computer connects to the Netlogon service on a domain controller (DC) in order to authenticate a user, the Netlogon services challenge each other to determine whether they both have a valid computer account. This allows a secure communication channel to be established for logon purposes.

In order for a Windows computer to join a domain, the computer must have a computer account in Active Directory. [<4>](#)

The computer name (for example RedmondDc1\$) is used as the Lightweight Directory Access Protocol (LDAP) relative distinguished name (RDN). Active Directory suggests the pre-Windows 2000 name that uses the first 15 bytes of the RDN. The administrator can change the pre-Windows 2000 name at any time.

The DNS name for a host is called a full computer name and is a DNS fully qualified domain name (FQDN). The full computer name is a concatenation of the computer name (the first 15 bytes of the SAM account name of the computer account without the "\$" character) and the primary DNS suffix (the DNS domain name of the domain in which the computer account exists). It is listed on the Computer Name tab in the System Properties dialog box in Control Panel.

By default, the primary DNS suffix portion of the FQDN for a computer must be the same as the name of the Active Directory domain where the computer is located. To allow different primary DNS suffixes, a domain administrator may create a restricted list of allowed suffixes by creating the **msDS-AllowedDNSSuffixes** attribute in the domain object container. This attribute is created and managed by the domain administrator by using Active Directory Service Interfaces (ADSI) or LDAP.

The SPN is a multivalue attribute. It is usually built from the DNS name of the host. The SPN is used in the process of mutual authentication between the client and the server hosting a particular service. The client finds a computer account based on the SPN of the service to which it is trying to connect. The SPN can be modified by members of the Domain Admins group.

2.3 Security Identifiers (SIDs)

The security identifier (SID) is an account identifier. The SID is variable in length and encapsulates the hierarchical notion of issuer and identifier. It consists of a 6-byte *identifier authority* field that is followed by one to fourteen 32-bit *subauthority* values and ends in a single 32-bit *relative identifier (RID)*. For example, a two-subauthority SID appears as shown in the following figure.

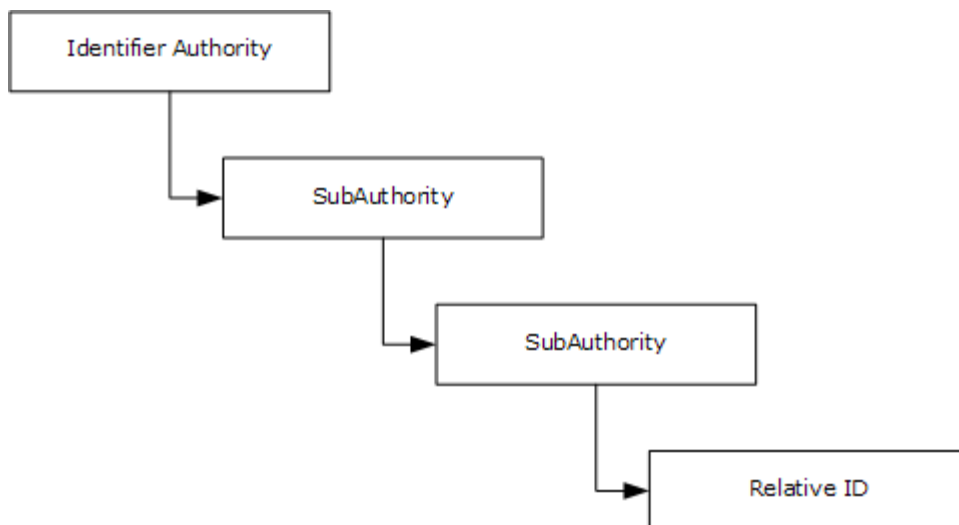


Figure 1: Windows security session identifier via subauthorities

When displayed textually, the accepted form is the following:

S-1-<identifier authority>--<sub1>--<sub2>--...-<subn>--<rid>

where *S* and *1* are literal strings, *identifier authority* is the 6-byte value, *sub1* through *subn* are the subauthority values, and *rid* is the RID.

The original concept of the SID called out each level of the hierarchy. Each layer included a new subauthority, and an enterprise could lay out arbitrarily complicated hierarchies of issuing authorities. Each layer could, in turn, create additional authorities beneath it. In reality, this system created a lot of overhead for setup and deployment and made the management model group even more complicated. The notion of arbitrary depth identities did not survive the early stages of Microsoft Windows® development, although the structure was already too deeply ingrained to be removed.

In practice, two SID patterns developed. For built-in, predefined identities, the hierarchy was compressed to a depth of two or three subauthorities. For real identities of other principals, the identifier authority was set to five, and the set of subauthorities was set to four.

Whenever a new issuing authority under Windows is created (for example, a new machine deployed or a domain created), it is assigned a SID with 5 (an arbitrary value) as the *identifier authority*; a fixed value of 21 is used as a unique value to root this set of subauthorities, and a 96-bit random number is created and parceled out to the three subauthorities with each subauthority that receives a 32-bit chunk.

Windows allocates RIDs starting at 1,000; RIDs having a value less than 1,000 are considered reserved and are used for special accounts. For example, all Windows accounts with a RID of 500 are considered built-in Administrator accounts in their respective issuing authorities.

Thus, a SID that is associated with an account appears as depicted in the following figure.

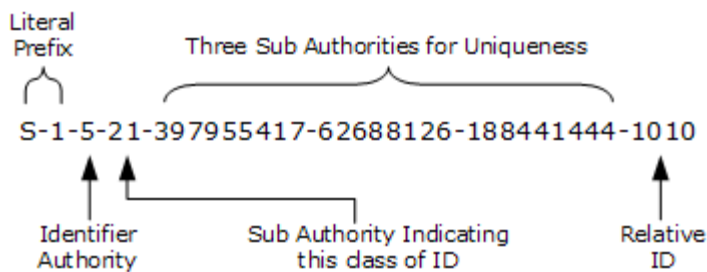


Figure 2: SID with account association

For most uses, the SID can be treated as a single long identifier for an account. By the time a specific SID is associated with a resource or logged in a file, it is effectively just a single entity. For some cases, however, it should conceptually be treated as two values: a value that indicates the issuing authority and an identifier relative to that authority. Sending a series of SIDs, all from the same issuer, is one example: the list can easily be compressed to be the issuer portion and the list of IDs relative to that issuer.

It is the responsibility of the issuing authority to preserve the uniqueness of the SIDs, which implies that the issuer must not issue the same RID more than one time. A trivial approach to this entails allocating RIDs sequentially. More complicated schemes are certainly possible; Active Directory uses a multimaster approach that allocates RIDs in blocks. It is possible for an issuing authority to run out of RIDs; therefore, the issuing authority must take care to handle this situation correctly. Typically, that authority must be retired.

Windows supports the concept of groups with much the same mechanisms as individual accounts. Each group has a name, just as the accounts have names. Each group also has an associated SID.

User accounts and groups share the same SID and namespaces. Users and groups cannot have the same name on a Windows-based system nor can the SID for a group and a user be the same.

For access control, Windows makes no distinction whether a SID is assigned to a group or an account. Changing the name of a user, computer, or domain does not change the underlying SID for that account, and an administrator cannot modify the SID for that account. Administrators cannot modify the SID for an account in Microsoft Windows NT® operating system, and there is generally no need to know the SID that is assigned to a particular account. SIDs are primarily intended to be used internally by the operating system to ensure that accounts are uniquely identified in the system.

2.4 Groups

A group is a collection of user accounts, computer accounts, and other groups that can be managed as a single unit from a security perspective. Groups can be based either in Active Directory or local to a particular computer.

Microsoft Windows® Server operating system has several built-in accounts and security groups that are preconfigured with the appropriate rights and permissions to perform specific tasks. [<5>](#) Active Directory provides two types of administrative responsibilities: service administrators are responsible for maintaining and delivering the directory service, including DC management and directory service configuration; and data administrators are responsible for maintaining the data that is stored in the directory service and on servers and workstations that are domain members.

It is important to understand which default accounts and groups are service administrators. Service administration accounts and groups have the most widespread power in the network environment and require the most protection.

2.4.1 Group Types

Starting with Microsoft Windows® 2000 operating system, Microsoft Windows® provides two types of groups:

- Security groups: These groups can contain members and can be granted permissions in order to control access to network resources. Security groups can contain users, other groups, and even computers.
- Distribution groups: These groups are used for nonsecurity functions, such as grouping users together to send e-mail messages. Unlike security groups, these groups cannot be used to control access to network resources.

Microsoft Windows NT® Server operating system has only one type of group, which is equivalent to security groups in Microsoft Windows® 2000 Server operating system, Windows Server® 2003 operating system, and Windows Server® 2008 operating system.

Starting with Windows 2000 Server, groups can be created by using the Active Directory Users and Computers console, and groups are stored as group objects in the Active Directory directory service. In Microsoft Windows NT® operating system, groups can be created by using User Manager for Domains, and groups are stored in the SAM database. Users can belong to multiple groups at the same time. A group does not actually contain its member user accounts; it is just a list of user accounts.

2.4.2 Group Scope

The scope of a group can be local or global depending on the portion of the network for which the group can be granted rights and permissions. Beginning with Microsoft Windows® 2000 operating system, Microsoft Windows® provides three levels of scope for security groups:

- Universal groups: These groups can contain members for any domain and be granted permissions to resources in any domain in a specific Active Directory forest. An Active Directory forest is a collection of one or more Active Directory domains that share a common logical structure, directory schema, and network configuration, as well as automatic two-way transitive trust relationships. Each forest is a single instance of the directory and defines a security boundary. Universal groups can contain user accounts, global groups, and universal groups from any domain in the current forest. An administrator can create a universal group only when the domain is in native mode and not in mixed mode as defined in section [2.4.3](#).
- Global groups: These groups can contain members only from their own domain but can be granted permissions to resources in any trusting domain. When the domain is in native mode, global groups can contain user accounts and global groups from the same domain. When the domain is in mixed mode, these groups can contain only user accounts.
- Domain local groups: These groups can contain members from any trusted domain but can be granted permissions only to resources in their own domain. Unlike Microsoft Windows NT® operating system local groups, a domain local group can be granted permissions to resources on all servers (both the DCs and member servers) in its domain. When the domain is in mixed mode, domain local groups can contain user accounts and global groups from any trusted domain or forest. When the domain is in native mode, domain local groups can also contain domain local groups from their own domain and universal groups from within any domain in the forest.

Beginning with Microsoft Windows® 2000 Server operating system, for member servers and client computers, and also for Windows® XP operating system clients, a fourth scope of group called a local group can exist only within the local security database of the computer where it is created. Local groups in Windows 2000 Server, Windows Server® 2003 operating system, Windows

Server® 2008 operating system, Windows XP, and Windows Vista® operating system are similar to local groups in Windows NT. They can contain user accounts that are local to the computer and user accounts and global groups from their own domain. A local group can be granted permissions to resources only on the computer where it was created. The Local Users and Groups Microsoft Management Console (MMC) is used to create local groups on a computer.

Windows NT groups have only two levels of scope:

- Global groups: A global group can be granted permissions to resources in its own domain and to resources in trusting domains. A global group can contain user accounts only from its own domain. Global groups are created on Windows NT domain controllers and exist in the domain directory database.
- Local groups: A local group created with Microsoft Windows NT® Workstation operating system can be granted permissions only to resources on the computer where it was created. A local group created with Microsoft Windows NT® Server operating system domain controller can be granted permissions only to resources on the domain controllers of its own domain. A local group can contain user accounts and global groups both from its own domain and from trusted domains. Network administrators of enterprise-level Windows NT networks can use a resource-access strategy called AGLP (Accounts organized by placing them in Global groups, which are then placed in Local groups that have appropriate Permissions and rights assigned to them) to plan and implement local groups in their network.

Beginning with Windows 2000 Server, the scope of a group can be changed. For example, global groups that are not members of other global groups can be converted to universal groups. Domain local groups that do not contain other domain local groups can be converted to universal groups.

2.4.3 Nested Groups

Microsoft Windows® supports the concept of nesting groups, or adding groups to other groups. Nesting groups can help reduce the number of permissions that need to be individually assigned to users or groups.

The process of creating groups across domains involves the following steps:

1. The administrator in each domain creates global groups and adds user accounts that have the same resource requirements to the global groups.
2. A domain administrator creates a domain local group for each resource that exists within a domain, such as file shares or printers, and then adds the appropriate global groups from each domain to this domain local group.
3. A domain administrator assigns the appropriate permissions for the resources to the domain local group. Users in each global group receive the required permissions because their global group is a member of the domain local group.

Effectively nesting groups in a multidomain environment reduces network traffic between domains and simplifies administration in a domain tree. A domain tree is a collection of domains that are grouped together in hierarchical structures so that they can be administered as single logical unit.

When a domain is added to the domain tree, it becomes a child of the tree root domain. The domain to which a child domain is attached is called the parent domain. A child domain can contain its own child domain. The name of a child domain is combined with the name of its parent to form its own unique DNS name such as Corp.mycompany.msft. In this manner, a tree has a contiguous namespace.

The extent to which nesting can be used in a specific organization depends on which mode the DC was configured in the system. Domain controllers can be configured in two modes: mixed mode or native mode.

- **Mixed mode:** A DC that is configured to support a mixed environment with Microsoft Windows NT® 4.0 operating system, Microsoft Windows® 2000 operating system, Windows Server® 2003 operating system, and Windows Server® 2008 operating system domain controllers in the same domain.
- **Native mode:** A DC that is configured to support only mixed Microsoft Windows® 2000 Server operating system, Windows Server 2003, and Windows Server 2008 environments.

In mixed mode, only one type of nesting is available; global groups can be members of domain local groups. Universal groups do not exist in mixed mode. In native mode, multiple levels of nesting are available. The nesting rules for group memberships for Windows 2000 are summarized in the following table.

Group scope	Can contain	Can be a member of
Domain local group	User accounts and universal and global groups from any trusted domain. Domain local groups from the same domain.	Domain local groups in the same domain.
Global group	User accounts and global groups from the same domain.	Universal and domain local groups in any domain. Global groups in the same domain.
Universal group	User accounts and universal and global groups from any domain.	Universal or domain local groups in any domain.

2.5 Account Domains

Accounts are always created relative to some issuing authority, which is responsible for allocating and assigning the SID. For Microsoft Windows® issuing authorities, this is referred to as a *domain*. Domains come in two varieties, local and remote.

2.5.1 Local Domains

Every computer running Microsoft Windows® has a local domain; that is, it has an account database for accounts that are specific to that computer. Conceptually, this is an account database like any other with accounts, groups, SIDs, and so on. These are referred to as local accounts, local groups, and so on. Because computers typically do not trust each other for account information, these identities stay local to the computer on which they were created.

2.5.2 Remote Domains and Domain Controllers

With a remote domain, certain Microsoft Windows®-based servers can be configured to be *DCs*. A DC is a server that has made its account database available to other machines in a controlled manner. Starting with Microsoft Windows® 2000 operating system, [6](#) DC began supporting a database of more than just accounts, becoming a general-purpose directory. This is known as Active Directory.

Because the account database is typically distributed across multiple DCs, there can naturally be a mix of different versions of the individual servers. Active Directory has the notion of a *functional*

level, which serves as a version level for the entire directory. The functional level is managed by the administrator and the system itself.

A domain has built-in groups; these groups are defined by Microsoft and created within the domain during installation. For example, built-in groups include the Domain Users, Domain Computers, and Domain Admins groups. By default, the Domain Users group includes all users who are defined in the domain.

A DC accepts authentication requests on behalf of the machines that have chosen to trust it.

A DC can have peers within the domain. These peers are other servers that also have been configured to host this account database. Any server participating in the domain as a domain controller may or may not allow changes; the configuration is a choice of the administrator. <7>

When a change is allowed, the servers replicate the change so that all DCs have the same information.

2.5.3 Domain Membership

Domain membership is the state of trusting a third party (the DC) for identity and authentication information. Any system can conceivably be part of a domain. Microsoft Windows®-based systems can easily be configured to be part of a domain and trust their DC for many tasks. Also, certain configuration changes are made, such as accepting the domain as the authoritative source of time.

Windows-based systems can have local groups that include members from a domain. This allows the member system to manage its resources in the manner most relevant to it and not be completely dependent on the decisions of the domain administrator. A domain administrator can create a domain local group for each resource that exists within a domain, such as file shares or printers, and then add the appropriate global groups from each domain to this domain local group. The domain administrator then assigns the appropriate permissions for the resources to the domain local group.

Joining a domain ultimately distills down to establishing an account on the domain that represents the system joining the domain, and to setting the password (or key) for the account on both the domain and the actual system. In Windows, this process is encapsulated in a domain join function (NetJoinDomain). Several tools, such as WinBind, exist for non-Windows operating systems to join a Windows domain.

All Windows-based systems have a component that manages their relationship with their DC. This component, called *Netlogon*, maintains the keys that are necessary for ongoing authentication of the member system to the DC. It also creates a general-purpose channel to the Netlogon instance on the DC. <8>

This channel is used by various authentication protocol implementations to redirect an authentication request to (or augment their activities with) their instance on the DC.

2.5.4 Effect on Accounts

Microsoft Windows® domains have an effect on the way that accounts and groups work. Some of this is by convention, and some is by design.

By convention, when a Windows-based system is added to a domain, the domain administrators group is made a member of the local administrators group.

By design, groups have different scopes when domains are involved. Groups can be defined to be globally known and therefore usable by other domains or known only within the domain in which they are defined.

2.5.5 Globally Unique Identifiers (GUIDs)

In Microsoft Windows® programming and in Windows operating systems, a globally unique identifier (GUID), as specified in [\[RFC4122\]](#), is a 128-bit value that is a binary unique identifier (ID) for a specific entity. The term universally unique identifier (UUID) is sometimes used in Windows protocol specifications as a synonym for GUID.

2.5.5.1 Uniqueness

All GUIDs are assumed to be unique; however, it cannot be said that they **MUST** be unique because there is no mechanism to enforce that uniqueness. Some GUIDs are also unpredictable. [\[RFC4122\]](#) defines five versions of GUID, one of which, version 4, is unpredictable by design. Because a GUID includes a version number field, no GUID of one version could equal a GUID of a different version.

Sometimes GUIDs are generated at design time and remain constant throughout the life of a protocol, such as a GUID that identifies a remote procedure call (RPC) interface or one that identifies a particular Active Directory schema. Such GUID values when used in a protocol are typically listed in the protocol document. Other GUIDs are generated at runtime by the protocol implementation itself and are used to identify transitory things such as individual sessions, connections, transactions, and activities.

Some protocols use unpredictable GUIDs as self-authenticating identifiers or nonces. That is, the GUID value (for example, a GUID that represents a client ID) is kept secret by both parties to the protocol and is used as an identifier. However, because it is assumed to have significant entropy, it also serves as a high-entropy password. Alternatively, a random GUID can be used as a nonce, which is a number that is used only one time and is unpredictable by the attacker. A nonce is typically used for the purpose of preventing replay attacks.

2.5.5.2 Internal Structure

A GUID has substructure, which a GUID generator needs to be aware of. A protocol stack implementation that uses a GUID, unless the individual protocol specification explicitly states the contrary, **SHOULD** treat that GUID as an opaque single quantity to which only equality or inequality tests are applied. Such a protocol implementation **SHOULD NOT** make decisions based upon the substructure of the GUID.

[\[RFC4122\]](#) defines five versions of GUID and specifies how they are constructed. The known substructure is represented by the version and variant fields. The remaining fields (what that RFC calls time stamp and node) are what make the GUID unique.

In a version 1 GUID, these fields carry a value that is derived from the time and the computer's network node address, respectively. A side effect of a version 1 GUID is that it identifies the machine on which it was generated (barring replacement or transfer of its network interface card).

In a version 4 GUID, these fields have been replaced by random bits. In generating a version 4 GUID, an implementation **SHOULD** use a Federal Information Processing Standard (FIPS)-approved pseudo-random number generator (PRNG) (as specified in [\[FIPS140-2\]](#) or later), but any superior source of random bits (such as a true hardware PRNG) **MAY** be used instead. For more information about entropy sources, see [\[RFC4086\]](#). If a PRNG is used as the source of random bits, it takes a parameter called a *seed*. Although many experts in the field have studied, tested, and approved each FIPS-approved PRNG algorithm, no approved or nonapproved PRNG output is or can be more unique or less guessable than its seed. [<9><10>](#)

Any implementation of version 4 GUIDs **MAY** implement two types of GUID: one GUID for uniqueness only and another GUID for use as a nonce. These types are characterized as follows:

- **Uniqueness-only:** This GUID is not kept secret and it is not required that it be unguessable. Therefore, the PRNG that is used to generate a version 4 GUID needs to be seeded only with values that have a high probability of being unique for that machine and that session. Typically, such a seed is formed by the cryptographic hash of uniqueness values, such as the CPU ID, MAC address, time of day, processor tick count, system process table, and system state.
- **Nonce or authenticator:** This GUID must be kept secret to preserve its security value. The PRNG that is used to generate a version 4 GUID for this purpose needs to be seeded with values that have a high probability not only of being unique but also of being unguessable by an attacker. Typically, such a seed is formed by the cryptographic hash of unguessable values of high entropy, such as the output of a hardware True Random Bit Generator, the system state readable in kernel mode, the history of system state over a long runtime, the accumulated entropy from earlier operation of the system (retained in a place that the attacker cannot access), the time of arrival of hardware interrupts, or the history of mouse positions as it moves.

A secret GUID that is good enough for a nonce is also good for uniqueness. Any system that generates cryptographic keys needs a source of true random or pseudo-random bits that are unguessable enough for building those keys. Therefore, it is common practice for an implementer to provide only one type of version 4 GUID, which is a type that is good enough for a nonce; and to use cryptographic-quality random bits for generating that GUID—even when building a GUID that is used only for uniqueness. [<11>](#)

2.5.5.3 Quality of Random Bits

Each use of a version 4 GUID has a measure of quality:

- **Uniqueness:** The probability that some other system will happen to generate the same GUID (without any conscious attempt by an attacker to create that collision).
- **Nonce:** The probability that some conscious attacker will be able to guess the generated GUID.

An ideal GUID that uses N bits of randomness collides with some chosen value by accident (violating uniqueness) with a probability proportional to 2^{-N} and is guessable by an attacker with work proportional to 2^N . This quality of randomness is measured by entropy. The ideal case that is described earlier is said to represent N bits of entropy.

If N allegedly random bits actually contain $M < N$ bits of entropy, the probability of accidental collision is proportional to 2^{-M} and the work for an attacker is proportional to 2^M . This leads, in the case of GUIDs for uniqueness, to a higher-than-ideal probability of accidental collision. If such a collision occurs, the two different identified objects will have the same ID, possibly leading to confusion. In the case of GUIDs for use as nonces, the lower work by the attacker might result in a successful replay attack.

Neither of these flaws, if one occurs, changes any protocol that uses such a GUID. It might change the security claims that such a protocol might make, but not the state machine, packet sequence, or packet contents of the protocol. The same applies to GUIDs that are used for uniqueness. If two GUIDs that were supposed to be different are accidentally the same, then the protocol implementation is not changed, only the probability that a mistaken identity might occur.

Therefore, the implementer of a protocol is not required by the protocol specification to guarantee any quality of random bits. Nothing in the specification of any Microsoft Communications Protocol Program (MCP) protocol directs a conformant implementation to look for, much less detect, any use of low-quality random bits in GUID generation. This means that even a very low-entropy random bit stream can be used to generate GUIDs that will allow a protocol to interoperate and be indistinguishable from any other protocol implementation, no matter what the quality of random bits in that other implementation.

Of course, implementers who prefer to minimize the confusion that would result from nonunique GUIDs or the replay attacks that would result from guessable nonces are well advised to use the best-quality random bit sources that they can find.

3 Authentication

With a basic understanding of identity, the next natural question is, how is identity proven across the network?

3.1 Background

The following underlying concepts and historical background are the foundation for the authentication model of Microsoft Windows®.

3.1.1 Authentication Concepts

The purpose of authentication is for two communicating entities to establish the identity of one or both parties. It is presumed that the communication medium between the two entities is completely hostile, and that an attacker can inspect any message or tamper with any message. Tamper here means change, suppress, or replay. Protocols must be developed that allow the two entities to authenticate in such a harsh environment. Commonly, the two entities consist of a client and a server.

Authentication can take on several aspects. For example, authentication of the server may be sufficient. The use of Secure Sockets Layer (SSL) on the Internet is primarily centered on assuring the client of the identity of the server. For protected networks, client authentication may be sufficient, because the valuable resource lives on a single server, and the server really only needs to address the identity of the client.

On modern networks, however, proving the identity of both the client and the server has become of the highest importance. Clients need to be assured of the identity of a server to avoid divulging something important to a rogue server; servers must be assured of the identity of clients to avoid granting clients inappropriate access. This security concept is typically called *mutual authentication*.

Ultimately, authentication must be performed by using cryptographic operations of some form, such as encryption or signatures. There are two main types of encryption: *symmetric* and *asymmetric*. Symmetric encryption uses the same key to encrypt and decrypt a message. Asymmetric encryption uses one key to encrypt and uses a different key to decrypt; these keys are linked by mathematical requirements. Signatures can be implemented in a number of ways through keyed hashes and encrypted hashes.

3.1.2 Practice

In the early 1990s, John Linn, then of Digital Equipment Corporation, proposed that applications not be tied to specific security protocols. This proposal was the genesis of the Generic Security Service Application Programming Interface (GSS-API). This concept has driven the model of most authentication protocols that are intended for use within an application protocol. This concept is generally referred to as GSS style or the GSS model. Note that there have been a number of channel-based protocols, such as Secure Sockets Layer (SSL), Transport Layer Security (TLS), and Secure Shell (SSH), that are intended to be below the application protocol layer.

This approach, however, has led to a simplified form of interaction between the application protocol and the authentication protocol. In this model, the application protocol is responsible for ferrying discrete, opaque packets that are produced by the authentication protocol. The application has no visibility into the contents of the message; its responsibility is merely to carry them. These messages, which GSS specifications refer to as *tokens*, implement the authentication process.

The application in this model first calls the authentication protocol on the client. The client portion of the authentication protocol creates a token and returns it to the calling application. The application

then transmits that token to the server side of its connection, embedded within the application protocol. On the server side, the server application extracts the token and supplies it to the authentication protocol on the server side. The server-side authentication protocol can process the token and possibly generate a response or determine that authentication is complete. If another token is generated, the application must carry it back to the client, where the process continues.

This exchange of security tokens continues until one or both sides determine that authentication is complete. If authentication fails, the application should drop the connection and indicate the error. If it succeeds, the application can then be assured of the identity of the participants, as far as the underlying protocol can accomplish.

When authentication is complete, session-specific security services can be available. The application can then invoke the authentication protocol to sign or encrypt the messages that are sent as part of the application protocol. These operations are done in much the same way, where the application can indicate what portion of the message is to be encrypted, and then must include a per-message security token. By signing or encrypting the messages, or both, the application can obtain privacy, resistance to tampering of messages, and detection of messages dropped, suppressed, or replayed.

3.1.3 History

Microsoft Windows® networking has its roots in the LAN Manager (LM) network product. LM was designed for a time when client authentication was sufficient for most needs, and when computational capacity was exceeded by the algorithms common at the time. For example, exhaustively searching Data Encryption Standard (DES) keys was unthinkable by any but the most dedicated government resources. LM authentication used a straightforward challenge-response style of authentication and was sufficient for many customers for many years.

When Microsoft decided to adopt the Kerberos protocol for Windows and move away from NT LAN Manager (NTLM), it required a substantial change for a number of protocols. This process is still going on today. Rather than repeat the process when circumstances required a new or additional security protocol, Microsoft chose to insert a protocol, in this case, Simple and Protected GSS-API Negotiation (SPNEGO), to allow security protocol selection and extension.

3.2 Authentication Protocols

Several protocols are available for authentication in Microsoft Windows®, each with different strengths and weaknesses, different capabilities, and different uses within the product.

3.2.1 NT LAN Manager (NTLM)

NT LAN Manager (NTLM) is an ongoing extension to the original LAN Manager (LM) authentication protocol. NTLM is conceptually straightforward and only performs client authentication. NTLM has undergone some revision (known as NTLMv2), which incorporates additional information into the computation of the response; however, it still follows the same general message flow.

3.2.1.1 Description and Flow

NT LAN Manager (NTLM) messages are carried by an application protocol. The basic flow of the security tokens or messages is as follows.

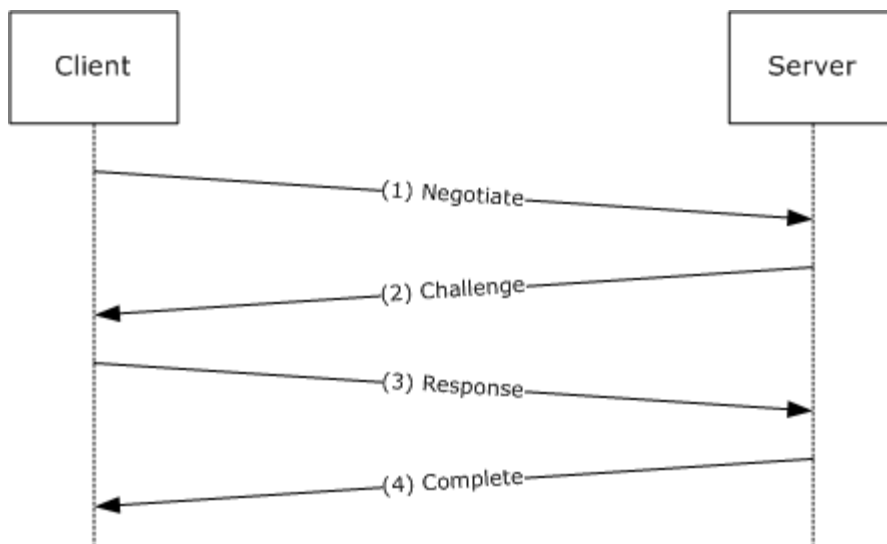


Figure 3: NTLM client/server session life cycle (basic)

1. The client sends an initial message to the server, advertising certain options or capabilities, such as cryptographic algorithm support.
2. The server creates a challenge, c , and returns the challenge and the options or capabilities that it can support to the client.
3. The client computes a function on the challenge, $resp = f(c, password)$, and sends the results to the server along with the user's textual name and domain.
4. The server looks up the user (by the name passed) and computes the same function, $f(c, user's\ password)$. If the result matches $resp$ (that is, what the client sent in step 3), the passwords are presumed to match, and the user is authenticated.

The details of the protocol can be found in the official documentation for NTLM; this explanation is for illustrative purposes only. For example, *password* (in step 3) is actually a hashed, derivative binary form of the actual textual password.

NTLM becomes more interesting when the domain scenario is factored in. Recall that for domains, the DC hosts the account database, and members of the domain do not have direct access to the account database anymore. In the domain scenario, the flow from client to server is the same, but there is a new interaction with the DC.

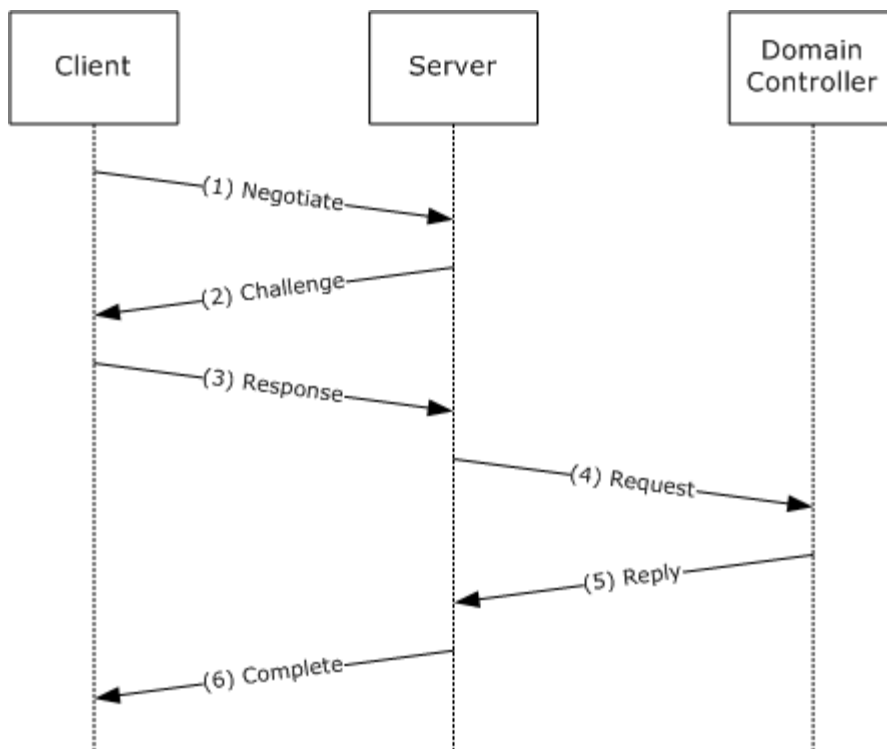


Figure 4: NTLM client/server session life cycle (with domain controller)

Here there are two additional messages. In the flow depicted in the preceding figure, the server adds the following:

1. Forward challenge, *c*, the response, *resp*, the user name, and the user domain from the client to the DC.
2. The DC looks up the account record for the user specified, computes $f(c, password)$, and compares this with the *resp* from the client. If the record matches, the DC can gather the user account and group memberships and send this information back to the server as the reply.
3. The server can respond to the client that authentication is complete.

The introduction of the server forwarding the authentication request to the DC is a powerful point of indirection. It allows different account authorities to be woven into a trust network that is completely invisible to the client. This can go on virtually ad infinitum; the server's DC can forward the request to other domains, and so on.

3.2.1.2 Notes

NTLM can never provide mutual authentication in all situations; the client can never be assured of the identity of the server in a general manner. The only time that NTLM can provide mutual authentication is when the client knows through out-of-band means that the user name used for authentication exists only on the target server.

This can constrain the authentication to at least within the extended set of trusted domains.

3.2.2 Kerberos

In 1993, Microsoft began working toward adopting the Kerberos protocol. Kerberos support for mutual authentication, transitive trust among authorities, extensibility, public inspection and review, and performance and caching, all pointed to this protocol as the authentication protocol for enterprise deployment for the foreseeable future. Microsoft Windows® 2000 operating system shipped in 1999 with the Kerberos protocol natively supported.

Because the Kerberos protocol is an Internet Engineering Task Force (IETF) protocol, a multitude of documents are available that describe how it works; see [\[RFC4120\]](#) as the canonical reference.^{<12>}

MIT also has a document that describes the ideas behind the Kerberos protocol in an approachable manner (see [Designing an Authentication System: A Dialogue in Four Scenes \[DIALOGUE\]](#)).

3.2.2.1 Description and Flow

The Kerberos protocol is based on a client and a server and a trusted third party called the Key Distribution Center (KDC). The KDC is associated with an account database and has a key that is shared with each client or server that it knows about. The management of the account database is explicitly outside the Kerberos protocol. It is presumed, however, that each client or server knows its own key through some method.

The Kerberos protocol is based on passing tickets from clients to servers; these tickets are originally crafted by the KDC that serves the domain in which the account lives. The ticket is useful because the receiving server knows that only the KDC can create that ticket; therefore, it is trustworthy. The whole process is bootstrapped by obtaining a ticket to the KDC, which is used to request further tickets. This ticket-granting ticket (TGT) is obtained from a KDC through a slightly different request.

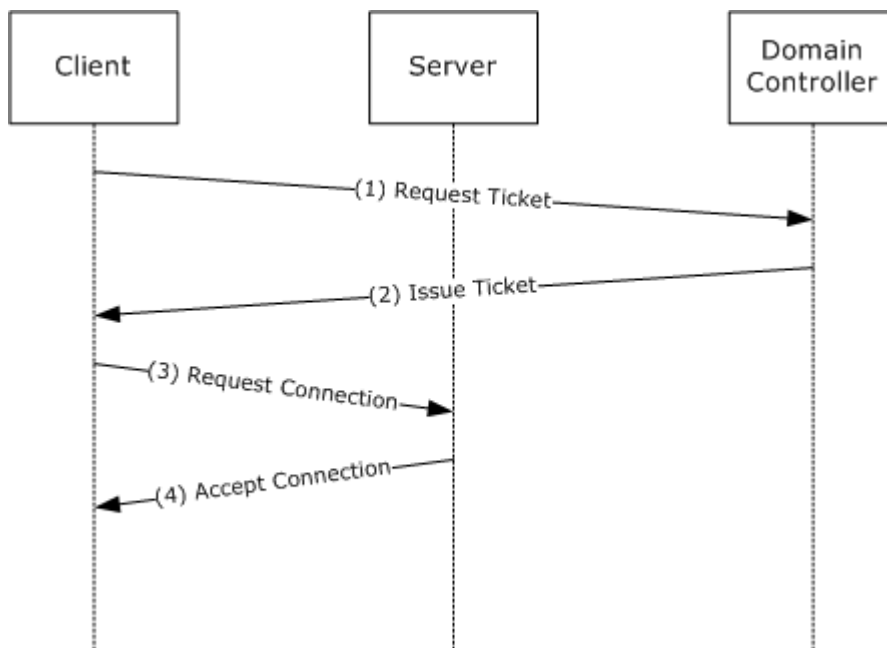


Figure 5: Windows security initialization sequence

The basic Kerberos exchange (assuming that the client already has the initial TGT) proceeds as follows:

1. The client issues a ticket request message to the KDC on the DC that requests a ticket to a named server. The ticket request message contains the TGT and cryptographic proof that the client can use that TGT.
2. The KDC on the DC looks up the server and issues a ticket to that server that is encrypted with the secret key of the server. The new ticket contains all the restrictions of the original TGT and any data that the TGT was carrying. The KDC creates a ticket reply message that contains the ticket and a copy of most of the ticket's contents encrypted for the client. The most important item in the ticket is a unique key that the client and the server are to use.
3. The client submits the ticket to the server in an authentication message. The message has the ticket from the KDC and has parts encrypted by using the unique key from step 2.
4. The server decodes the authentication message and finds the ticket. In the ticket, it finds the unique key from step 2. It can then decrypt and verify the rest of the authentication message. When decryption is successful, the server creates the authentication reply message, which it encrypts with the unique key. On receipt, the client verifies that the unique key is used.

3.2.2.2 Notes

Kerberos is strictly an authentication protocol; it is designed to convey only the identity of the principals on each side of the connection. The Kerberos protocol does contain extensibility points to allow extra vendor-defined information to be conveyed (along with the ticket) during authentication. This is expressed as the *auth_data* field in the Kerberos ticket.

Microsoft Windows® uses this *auth_data* field in the ticket to pass along the SID of the account and to group SIDs during authentication. The structure that is used for this behavior is termed the privilege attribute certificate (PAC). The PAC contains all the account's group memberships and is used to provide sufficient context on the server to make authorization decisions (for example, "can this user open this file?"). The encoding of the group information is publicly available from Microsoft. See [Utilizing the Windows 2000 Authorization Data in Kerberos Tickets for Access Control to Resources \[KERBPAC\]](#).

Although group membership information is a common use of the *auth_data* field, Microsoft also included additional profile and account management information, such as the location of the account's root directory. This design greatly aids certain enterprise deployment scenarios. This additional information is used for supporting interactive logon. These additions are available through the [Microsoft Protocol Technology Licensing Programs \[MSFT-LEGAL\]](#).

A strength of the Kerberos protocol is that it is not involved after the ticket is issued until the ticket expires and the client needs a new one. This behavior means that the client and server do not need to involve the KDC for every connection.

Much like other protocols, the Kerberos protocol can leverage more than one domain. If the server that is requested by a client is in a different domain, the KDC can return a TGT to that other domain. The client can then retry the request on a KDC in the other domain. This can span multiple domains as long as a trusting relationship exists among the domains.

One issue with a secret key system such as the Kerberos protocol is that any person who knows the key for a principal can create a ticket to that principal. Thus, even though the KDC is the normal creator of tickets, a person who knows the password for a principal could create and send a ticket to that principle.

For authentication only, this action is not a threat. The ticket created by the person is, in effect, fooling the principal with this action.

When the ticket includes authorization data that is respected and interpreted by something other than code running as that user, the potential for problems arises. In the Windows implementation, the authorization data results in a Windows *access token*, which is a system-provided object that encapsulates an account's identity, group memberships, and system privileges. An access token on Windows is used to make authorization decisions by the system (for example, validating access to a file).

A malicious user might construct a ticket to itself that contains a PAC indicating that the user is a member of a group (for example, the Administrators group) that it should not be. If processed naively, the system accepts this PAC and allows the user inappropriate access to the system. Windows compensates for this by involving the DC when a nonprivileged server receives a ticket. The Windows Kerberos implementation calls back to the DC through the Netlogon channel to verify the contents of the PAC.

3.2.3 Secure Sockets Layer (SSL) and Transport Layer Security (TLS)

SSL is a protocol first introduced by Netscape to allow browsers to authenticate servers. SSL went through a number of revisions, culminating in an adoption by the IETF and becoming the TLS protocol. For more information, see [\[RFC2246\]](#).

Although SSL and TLS are primarily used for authenticating servers and creating a secure pipe between the client and the server, they do allow for authenticating the client. At almost any point after the channel is established, the server can demand client authentication. The client signs a challenge from the server with its private, asymmetric key, and then sends both the signed data and the certificate for that key to the server.

In Microsoft Windows®, the certificate can be associated with an account in several ways, based on local policy and on what fields from the certificate are interesting to the administrator of the server or domain. Ultimately, however, the SSL/TLS implementation on the server calls up to the DC through the Netlogon channel and asks the SSL instance on the DC to determine what account is associated with this certificate.

The account is determined through a number of possible mappings, based on the fields present in the certificate. For example, *subjectAltName*, *commonName*, and other similar fields can be used to find the account in Active Directory. After the account is found, the account information (such as account SID and group membership SIDs) is returned to the client. Much like the NTLM case involving the domain, this can extend through arbitrary trust relationships. The format for this information is the same as the format of the PAC data from the Kerberos protocol.

3.2.4 Simple and Protected GSS-API Negotiation Mechanism (SPNEGO)

SPNEGO is an authentication protocol that actually does no authentication. Rather, it is an authentication protocol that allows secure negotiation among other security protocols when the client and the server might support more than one protocol. Microsoft Windows® uses SPNEGO instead of a specific protocol to allow for simpler substitutions of additional security protocols.

3.2.4.1 Description and Flow

SPNEGO fits into the system as a security protocol and shuttles the actual messages of the underlying security protocols back and forth between the client and the server. SPNEGO does not implement any specific security features but performs the important operation of enforcing policy among security protocols.

SPNEGO is defined as a GSS mechanism, which follows the general model of passing security tokens from one machine to another. SPNEGO is a simple protocol that passes a list of acceptable security

protocols (also called mechanisms) along with an optimistic mechanism token from one of these protocols. An optimistic mechanism token is the first context establishment token of the initiator's preferred mechanism embedded in the initial negotiation message, as specified in [\[RFC4178\]](#) section 4.2. The response from the server is what protocol has been selected and, optionally (if it matched the optimistic mechanism token), a response from that security protocol.

The SPNEGO module asks each available system protocol, in turn, if that protocol can potentially authenticate the server that is named by the component invoking SPNEGO. In the Microsoft Windows® implementation, the first protocol to respond positively is selected as the optimistic choice. After the protocol is selected, SPNEGO gets out of the way and lets the application protocol work directly with the selected authentication protocol.

3.2.4.2 Notes

In Microsoft Windows®, SPNEGO has the important task of deciding which protocol is valid for a particular connection. It does this by interpreting a number of input states and then adjusting the list of security protocols it offers the server. The SPNEGO implementation knows that NTLM cannot implement the same guarantees (due to legacy) that more modern protocols can. Therefore, NTLM is treated in a special manner on a Windows-based system. To decide the most appropriate protocol, the SPNEGO implementation takes the following information:

- User domain functional level, if known.
- Systemwide mutual authentication policy.
- Name of the server.

If the user is a member of a Microsoft Windows® 2000 operating system or later domain, SPNEGO knows that NTLM is a second choice and that at least the Kerberos protocol should be available. Therefore, if the Kerberos protocol indicates that it should be used (by returning an optimistic token), SPNEGO does not allow further downgrade to NTLM if the Kerberos authentication fails. Also, if the KDC cannot be contacted, and the user is a member of a Windows 2000 or later domain, the SPNEGO implementation fails, indicating that it cannot authenticate the connection. This failure is to prevent a class of security attacks.

3.3 Use of Authentication

The following section describes how authentication works in Microsoft Windows®.

3.3.1 General Model

Microsoft Windows® generally follows the GSS model of security. In the GSS model, application protocols try to be as agnostic as possible for the specific forms of authentication used in the session. In return, the security protocol (or mechanism) limits its specific behavior to some well-defined interfaces and communicates in opaque messages, which are referred to as *tokens* in the GSS documents.

Therefore, the application is responsible for calling the security runtime to secure its protocol. Conversely, the application does not need to be aware of any specifics of the security protocol that it has selected. This behavior leads to the somewhat surprising condition that an application protocol may not know which security protocol it is using in some conditions.

3.3.2 Known Idiosyncrasies

The following protocols have specific idiosyncratic behavior with regard to use of authentication protocols. In most cases, this is due to externally mandated behavior or certain legacy constraints.

3.3.2.1 Server Message Block (SMB) and Common Internet File System (CIFS)

Server Message Block (SMB) is the oldest general application protocol in Windows, and it predates the movement toward the GSS authentication model and has the NTLM built into its session establishment messages. SMB uses NTLM when the server is not compatible with Microsoft Windows® 2000 operating system or later, or when SPNEGO has determined that the server is not expected to use a protocol other than NTLM. In this case, the SMB client extracts the response values from the NTLM response token and directly supplies these in the message that is used to authenticate to the SMB server.

SMB uses a Microsoft Windows® extension to SPNEGO in which the server can create a hint to the client. The hint is not critical to the success of the protocol. The hint can help the client determine the best security protocol to use with the server. The hint allows the client to determine the following:

- The set of mechanisms available on the server.
- The claimed domain of the server.

4 Authorization

After an identity is suitably authenticated, the natural next step is to use that identity to authorize access to a resource. Microsoft Windows® has a very expressive authorization model available for applications and system components to use for making authorization decisions.

Windows was originally designed to meet the requirements of the C2 level of the Trusted Computer System Evaluation Criteria (TCSEC). The TCSEC program has since been supplanted by profiles written under the Common Criteria for Information Technology Security Evaluation, such as the Controlled Access Protection Profile. These profiles and related information can be found in section [6](#).

The C2 requirements (and later the CAPP requirements) for authorization are centered on *discretionary* access control. For discretionary access control, the owner of a particular resource (or a delegate of the owner) determines what access others should have. This is in contrast to *mandatory* access control schemes in which another party maintains control over the resource regardless of the expectations of the owner.

4.1 Resource Managers

Microsoft Windows® meets these requirements for discretionary access control by providing a single access evaluation routine that any number of *resource managers* can invoke. Many resource managers—including the file system, registry, Active Directory, and operating system constructs such as processes—exist in a Windows-based system. Even though these resource managers control very different objects, they share a common method for controlling access.

In the Windows authorization model, a resource manager is the code or component that implements one or more object types. The NTFS file system is a resource manager that implements files and directories; the Windows registry is a resource manager that implements keys. To participate in the authorization scheme, the resource manager is required to maintain a *security descriptor* with each object that is protected. The resource manager merely needs to maintain the storage for the security descriptor and is not required to understand the contents.

The Windows Security Overview also distinguishes between ordinary objects in the resource manager and *containers* exposed by the resource manager. In the file system example, files are objects, and directories are containers. This distinction is important during the creation of new objects.

4.2 Security Descriptors

The security descriptor is a collection of four main elements. The *owner* field is a SID that specifies the owner of the resource. The *group* field specifies the group associated with the resource. The *group* field is not evaluated by Microsoft Windows® components, and it exists for POSIX compatibility. The *DACL* field specifies the discretionary access control list, and the *SACL* field specifies the system access control list (SACL).

When associated with a resource, the security descriptor is intended to be opaque. The resource manager should never be required to examine the contents of the security descriptor. The security descriptor fields can be used by the resource manager for other purposes, however. For example, the file system can implement a storage quota system by using the owner field associate resources consumed with an owner for billing.

4.3 Discretionary Access Control Lists (DACLS) and Access Control Entries (ACEs)

Discretionary access control lists (DACLS, but often shortened to ACLs) form the primary means by which authorization is determined. An ACL is conceptually a list of <account, access-rights> pairs, although they are significantly richer than that.

Each pair in the ACL is termed an access control entry (ACE). Each ACE has additional modifiers that are primarily for use during inheritance. There are also several different kinds of ACEs for representing both access to a single object (such as a file) and access to an object with multiple properties (such as an object in Active Directory).

The ACE contains the SID of the account to which the ACE pertains. The SID can be for a user or a group.

Microsoft Windows® supports both positive ACEs, which grant or allow access rights to a particular account, and negative ACEs, which deny access rights to a particular account. This allows a resource owner to specify, for example, *grant read-access to group Y, except for user Z*.

DACLs can be configured at the discretion of any account that possesses the appropriate permissions to modify the configuration, including Take Ownership, Change Permissions, or Full Control permissions. DACLS consist of the following elements.

Header: Metadata pertaining to the ACEs associated with the DACL.

SID (user): The SID of the owner of the object.

SID (group): The SID of the built-in Administrators or Domain Admins group if the account that owns the object is a member of either of these groups.

Access control entry (ACE): A DACL can contain several ACEs. The following list identifies and describes these terms:

Explicit allow ACE: An ACE applied directly to the resource that grants access. An explicit allow will always override an inherited deny but will always be overridden by explicit deny ACEs.

Explicit deny ACE: An ACE applied directly to the resource that denies access. An explicit deny will always override all other permissions.

Generic deny ACE: An ACE that denies access to an account or security group based on the user or group SID. A generic deny ACE can be inherited from the object's parent or assigned directly to the object. The generic deny ACE is specific to the object and child objects of the same class, based on the security settings defined in the object class in the schema.

Generic allow ACE: An ACE that allows access to an account or security group based on the user or group SID. A generic allow ACEs can be inherited from the object's parent or assigned directly to the object. The generic allow ACE is specific to the object and child objects of the same class, based on the security settings defined in the object class in the schema.

Inherited deny ACE: An ACE inherited from the resource's parent object. An inherited deny ACE overrides an inherited allow permission, but is overridden by an explicit allow.

Inherited allow ACE: An ACE inherited from the resource's parent object.

Object-specific deny ACE: An ACE used within Active Directory that does one of the following:

- denies access to a property on the Active Directory object
- denies access to a property set on the Active Directory object
- limits the ACE inheritance to a specified type of child object based on the SID(s) of the child or children.

The object-specific deny ACE can be inherited from the object's parent or assigned directly to the object. Also, the object-specific deny ACE applies to specific classes of child objects.

Object-specific allow ACEs: An ACE used within Active Directory that does one of the following:

- Allows access to a property on the Active Directory object
- Allows access to a property set on the Active Directory object
- limit the ACE inheritance to a specified type of child object based on SID(s) of the child or children.

The object-specific allow ACE can be inherited from the object's parent, or assigned directly to the object/ Also, the object-specific allow ACE applies to specific classes of child objects.

When access is requested to an Active Directory object, the Local Security Authority (LSA) compares the access token of the account that is requesting access to the object to the DACL. The security subsystem checks the object's DACL, looking for ACEs that apply to the user and group SIDs referenced in the user's access token. The security subsystem then steps through the DACL until it finds any ACEs that allow or deny access to the user or to one of the user's groups. The subsystem does this by first examining ACEs that have been explicitly assigned to the object and then examining ones that have been inherited by the object. The following illustration shows the important parts of an access token and a DACL when a request is evaluated.

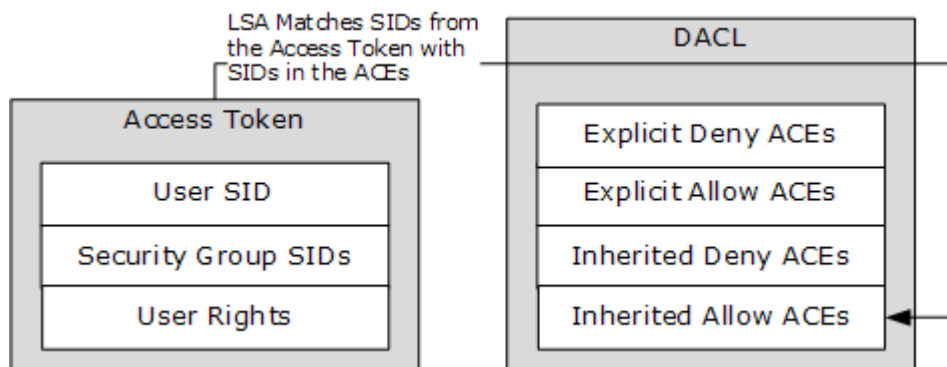


Figure 6: Evaluation process for access tokens against a DACL

If an explicit deny is found, access is denied. Explicit deny ACEs are always applied, even if conflicting allow ACEs exist. Explicit allow ACEs are examined, as are inherited deny and allow ACEs. The ACEs that apply to the user are accumulated. Inherited deny ACEs overrule inherited allow ACEs

but are overruled themselves by explicit allow permissions. If none of the user SIDS or group SIDs in the access token match the DACL, the user is denied access implicitly.

In Windows, a security principal's level of access to files and folders is determined by NTFS file system and share permissions. These permissions are discretionary; anyone with ownership of a file or folder, Change permissions, or Full Control permissions can assign access control at their discretion. When newly installed, Windows assigns default permission structures to operating system files and folders, but a user might need to alter these permissions to meet specific security requirements.

When a user attempts to access a file or folder on an NTFS partition, the user's access token is compared with the DACL of the file or folder. If no ACEs correspond to a SID in the user's access token, the user is implicitly denied access to the resource. If ACEs correspond to the user's access token, the ACEs are applied in the following order.

1. Explicit deny
2. Explicit allow
3. Inherited deny
4. Inherited allow

ACEs that apply to the user are cumulative, meaning that the user will receive the sum of the ACEs that apply to his or her user account and groups of which the user is a member. For example, if an ACL contains two allow ACEs that apply to the user, one for Read access and the other for Write access, the user will receive Read and Write access.

4.4 Access Rights

Different resource managers and resource types have different access rights. Files may have read and write access, but processes have entirely different rights such as terminate. However, all resource managers use the same formats for encoding access rights in the ACEs. This is done by allowing the resource managers to define their own specific access rights.

Microsoft Windows® accomplishes this by partitioning the access rights space. All access rights are encoded into a single, 32-bit value in the ACE. The most significant 16 bits are considered standard access rights and are common across all resource managers. These rights include Delete access, Generic-Read access, and other similar rights. These rights are either expected of all resource managers (such as Delete) or are used in a way that allows programs to work with multiple resource managers in a similar manner.

The least significant 16 bits are termed object-specific and are meaningful only to the resource manager that defines them. Thus the file system may define that bit 1 indicates the capability to read the file and that bit 2 indicates the capability to write the file, whereas the registry may define bit 1 to be enumerate subkeys and bit 2 to be read a key's value.

4.5 Authorization

Microsoft Windows® has a single method in the system for determining access. In that way, the results are always predictable and consistent. The process is as follows.

To determine access, the calling resource manager supplies the security descriptor (which contains the ACL) with the identity of the user and all the groups of which the user is a member, and the access requested by the user. For this example, the following values are used.

```
Security Descriptor: Owner: U1, DACL: <<U2, Read>, <G1, Read>,
<G2, Write>>
Identity: <U1, G2>
Access Request: Write
```

In this example, the security descriptor has an ACL that grants U2 Read access, G1 Read access, and G2 Write access. The identity of the user making the request is U1, and that user is a member of the group G2 as well. The request is for Write access.

When processing this request, Windows iterates through the entries in the ACL, testing against the identity. If the identity in the ACE matches one of the identities of the user, the ACE is examined further. In this example, the first two ACEs do not match any identity, and so they are skipped. The third ACE applies (G2 matches), and then the granted access rights are compared against the access request. They match, and the user is therefore granted access.

As noted earlier, multiple access rights are encoded together, so the access request could be for both Read access and Write access. In the preceding example, access would be denied because G2 was granted only Write access.

All the requested rights do not have to be granted by a single ACE. Consider the following example.

```
Security Descriptor: Owner:U1, DACL:<<U2,Read>,<G1,Read>,<G2,Write>>
Identity:<U1,G1,G2>
Access Request: Read,Write
```

The processing would be as follows.

The first ACE does not match, and so it is skipped. The second ACE now does match and is therefore examined further. The granted access is removed from the access request, in this case, Read. There are still values left in the access request, so processing continues. The third ACE matches (on G2) and grants Write access. The granted access, Write, is removed from the access request, but now there are no remaining access requests. The access is granted, and processing stops.

4.6 Inheritance

The Microsoft Windows® authorization model supports a concept of inheritance by which new objects can inherit one or more ACEs from their parent container. In practice, this allows an administrator to establish default security on, for example, a directory, and all new files that are created in that directory receive a preset ACL. Although the owner of the file can still override that ACL and establish its own, if nothing is done (through the premise of discretionary access control), the default is as the administrator wants.

One attribute that can be applied to ACEs is the *Object-Inherit* flag. This flag indicates that when a new object is created, this ACE should be carried forward to the security descriptor of the new object. An additional flag, *Container-Inherit*, indicates that new containers created under this container should receive this ACE. For the file system, this allows different default ACLs for directories as opposed to files.

4.7 System Access Control Lists

A system access control list (SACL) enables administrators to log attempts to access a secured object. Like a DACL, an SACL is a list of ACEs. Each ACE specifies the types of access attempts by a specified account that cause the system to generate a record in the security event log. An ACE in an SACL can generate audit records when an access attempt fails, when it succeeds, or both.

4.8 Windows Integrity Mechanism

The Microsoft Windows® integrity mechanism extends the security architecture by defining a new access control entry (ACE) type to represent an integrity level in an object's security descriptor. The new ACE represents the object integrity level. An integrity level is also assigned to the security access token when the access token is initialized. The integrity level in the access token represents a subject integrity level. The integrity level in the access token is compared against the integrity level in the security descriptor when the security reference monitor performs an access check. Windows Vista® operating system uses the AccessCheck function to determine what access rights are allowed to a securable object. Windows restricts the allowed access rights depending on whether the subject's integrity level is higher or lower than the object, and depending on the integrity policy flags in the new access control ACE. The security subsystem implements the integrity level as a mandatory label to distinguish it from the discretionary access under user control that ACLs provide.

Note: The mandatory integrity labels are not evaluated as part of the discretionary access control by the AccessCheck algorithm ([\[MS-DTYP\]](#) section 2.5.3.2). These SIDs are not set in the token with the SE_GROUP_ENABLED attributes, which means that even though these SIDs are added to the SIDS_AND_ATTRIBUTES array in the token, they are not evaluated as part of sidintoken.

5 Impersonation

In distributed systems, it is typical for one server to call another server to accomplish a task for a client. This functionality is called **impersonation**. To handle these requests for a client, the server must be given the authority to do so. The ability to call other servers while impersonating the original client is called **delegation**.

Through impersonation, a thread runs in a security context that is different from the context of the process that owns the thread. When a server thread runs in the security context of the client, it uses an access token that represents the client credentials in order to obtain access to the objects to which the client has access. This provides the ability for a thread to run by using different security information from the process that owns the thread. Typically, a thread in a server application impersonates a client. This impersonation allows the server thread to act for that client in order to access objects on the server or validate access to the client objects.

The following diagram shows the impersonation process. A client makes a request to server A. If server A must query server B to complete the request, server A impersonates the client security context and makes the request to server B for the client. Server B uses the security context of the original client, instead of the security identity for server A, to determine whether to complete the task.

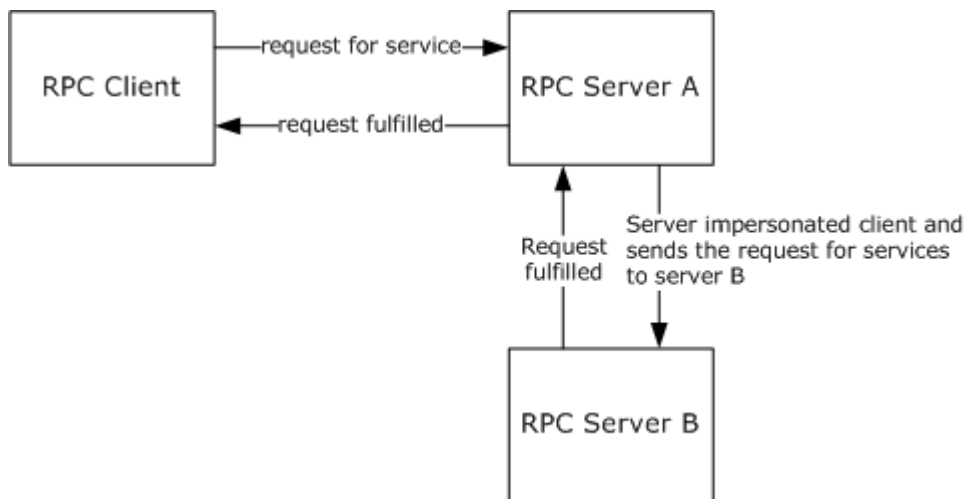


Figure 7: Impersonation process

When delegation is used, a server that is impersonating a client can call another server and can make network calls by using the credentials of the client. From the perspective of the second server, requests that come from the first server are indistinguishable from requests that come from the client. Not all security providers support delegation. Microsoft Windows® provides only one security provider that supports delegation: the Kerberos protocol.

Delegation must be implemented with caution due to the privileges that the client gives the server during an RPC. To address this, the Kerberos protocol allows calls that use the impersonation level of delegation only if mutual authentication is requested. Domain administrators can limit the computers to which calls with delegation impersonation level are made to prevent unsuspecting clients from making calls to servers that can abuse their credentials.

Calls that use `ncalrpc` [\[MSDN-NCALRPC\]](#) are an exception to the delegation rule. When these calls are made, the server gets delegation rights, even if an impersonation level of impersonate is specified. That is, a server can call other servers on behalf of the client. This works for one remote

call only. For example, if client A calls local server LB using **ncalrpc**, local server LB can impersonate and call remote server RB. Remote server RB can act for client A, but only on the remote computer on which remote server RB is running. Local server LB cannot make another network call to remote computer C unless LB specifies an impersonation level of delegate when it calls RB.

A primary use of impersonation is to perform access checks against the client identity. Using the client identity for access checks can cause access to be either restricted or expanded, depending on what the client has permission to do. For example, a file server might have files that contain confidential information and each of these files is protected by an ACL. To help prevent a client from obtaining unauthorized access to information in these files, the server can impersonate the client before accessing the files.

5.1 Cloaking

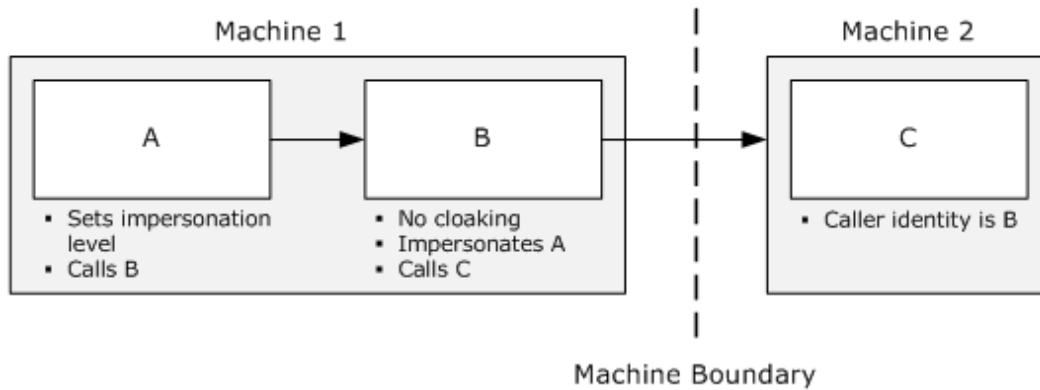
There are two important security considerations regarding [impersonation](#) and delegation:

- What should the server be allowed to do when acting for the client?
- What identity is presented by the server when it calls other servers for a client?

In Microsoft Windows®, the Component Object Model (COM) provides the functionality that is explained here. The client can set an [impersonation level](#) that determines the extent to which the server can act as the client. If the client grants enough authority to the server, the server can impersonate (pretend to be) the client. When the server impersonates the client, it is given access to only those objects or resources that the client has permission to use. The server, acting as a client, can also enable [cloaking](#) in order to mask its own identity and project the client identity in calls to other COM components.

The following figure illustrates impersonation without and with cloaking. A and B represent two processes running on machine 1, and C represents a process that is running on machine 2. Process A calls B, and B calls C. Client A sets the impersonation level. B sets the cloaking capability. If A sets an impersonation level that permits impersonation, B can impersonate A when calling C on A's behalf. The identity that is presented to process C is either A's identity or B's identity, depending on whether cloaking was enabled by B. If cloaking is enabled, the identity that is presented to process C is A. If cloaking is not enabled, B's identity is presented to C.

A) Impersonation without Cloaking



B) Impersonation with Cloaking

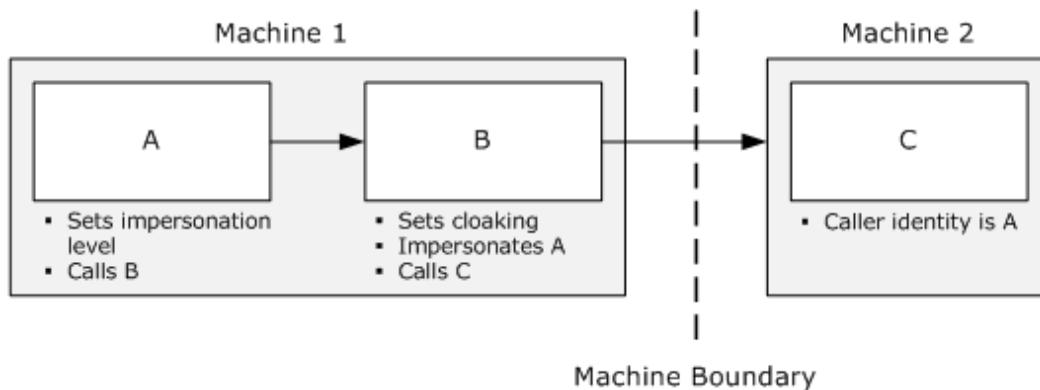


Figure 8: Impersonation without and with cloaking

5.2 Impersonation Tokens

Access tokens are objects that describe the security context of a process or a thread. They provide information that includes the identity of a user account and a subset of the privileges that are available to the user account. Every process has a *primary access token* that describes the security context of the user account that is associated with the process.

By default, the system uses the primary token when a thread of the process interacts with a securable object. However, when a thread impersonates a client, the impersonating thread has both a primary access token and an impersonation token. The impersonation token represents the security context of the client, and this access token is the one that is used for access checks during impersonation. When impersonation is complete, the thread reverts to using only the primary access token.

5.3 Impersonation Levels

Microsoft Windows® provides various degrees of impersonation through impersonation levels, which indicate how much authority is given to the server when it is impersonating the client.

Currently, there are four impersonation levels available: anonymous, identify, impersonate, and delegate. [<13>](#) The following list briefly identifies and describes each impersonation level:

- **Anonymous level (RPC_C_IMP_LEVEL_ANONYMOUS):** The client is anonymous to the server. The server process can impersonate the client, but the impersonation token does not contain any information about the client. This level is supported only over the local interprocess communication transport. All other transports silently promote this level to identify.
- **Identify level (RPC_C_IMP_LEVEL_IDENTIFY):** The system default level. The server can obtain the identity of the client, and the server can impersonate the client in order to do ACL checks.
- **Impersonate level (RPC_C_IMP_LEVEL_IMPERSONATE):** The server can impersonate the security context of the client while acting for the client. The server can access local resources as the client. If the server is local, it can access network resources as the client. If the server is remote, it can access only resources that are on the same machine as the server.
- **Delegate level (RPC_C_IMP_LEVEL_DELEGATE):** The most powerful impersonation level. When this level is selected, the server (whether local or remote) can impersonate the security context of the client while acting on behalf of the client. During impersonation, the client credentials (both local and network) can be passed to any number of machines. [<14>](#) This level is supported beginning with Windows 2000

For impersonation to work at the delegate level, the following requirements must be met:

- The client must set the impersonation level to `RPC_C_IMP_LEVEL_DELEGATE`.
- The client account must not be marked "Account is sensitive and cannot be delegated" in the Active Directory directory service.
- The server account must be marked with the "Trusted for delegation" attribute in Active Directory.
- The computers that host the client, the server, and any "downstream" servers must all be running Windows 2000 in a Windows 2000 domain.

By choosing the impersonation level, the client tells the server how far it can go in impersonating the client. The client sets the impersonation level on the proxy that it uses to communicate with the server.

5.4 Setting the Impersonation Level

There are two ways to set the impersonation level:

- The client can set the impersonation level processwide through a call to [CoInitializeSecurity](#).
- A client can set proxy-level security on an interface of a remote object through a call to [IClientSecurity::SetBlanket](#) (or the helper function [CoSetProxyBlanket](#)).

The impersonation level is set by passing an appropriate `RPC_C_IMP_LEVEL_xxx` value to [CoInitializeSecurity](#) or [CoSetProxyBlanket](#) through the `dwImpLevel` parameter.

Different authentication services support delegate-level impersonation to a different extent. For example, NTLMSSP in Microsoft Windows® 2000 operating system supports cross-thread and cross-process delegate-level impersonation but not cross-machine. However, the Kerberos protocol (implemented by Windows 2000) supports delegate-level impersonation across machine boundaries, whereas SChannel does not support any impersonation at the delegate level. If there is a proxy at

the impersonate level and the implementer wants to set the impersonation level to delegate, [IClientSecurity::SetBlanket](#) SHOULD be called by using the default constants for every parameter except the impersonation level. COM chooses NTLM locally and the Kerberos protocol remotely (when the Kerberos protocol is supported).

5.5 Windows API Impersonation Functions

The Microsoft Windows® API provides the following functions to begin an impersonation:

- A Dynamic Data Exchange (DDE) server application can call the [DdeImpersonateClient](#) function to impersonate a client.
- A named pipe server can call the [ImpersonateNamedPipeClient](#) function.
- The [ImpersonateLoggedOnUser](#) function can be called to impersonate the security context of an [access token](#) for a logged-on user.
- The [ImpersonateSelf](#) function enables a thread to generate a copy of its own access token. This is useful when an application needs to change the security context of a single thread. For example, sometimes only one thread of a process needs to enable a [privilege](#).
- The [SetThreadToken](#) function can be called to cause the target thread to run in the security context of a specified [impersonation token](#).
- An RPC server application can call the [RpcImpersonateClient](#) function to impersonate a client.
- A [security package](#) or application server can call the [ImpersonateSecurityContext](#) function to impersonate a client.

For most of these impersonations, the impersonating thread can revert to its own security context by calling the [RevertToSelf](#) function. The exception is the RPC impersonation, in which the RPC server application calls [RpcRevertToSelf](#) or [RpcRevertToSelfEx](#) to revert to its own security context.

For more information about the Windows API impersonation functions, visit the [MSDN home page](#).

6 References

The following books include useful content that covers aspects of the Windows Security Model in greater depth:

- Brown, Keith, "Programming Windows Security", Addison-Wesley Professional, 2000, ISBN 0201604426.
- Howard, Michael and LeBlanc, David, "Writing Secure Code", Microsoft Press, 2002, ISBN 0735617228.
- Russinovich, Mark E. and Solomon, David A., "Microsoft Windows Internals, 4th ed.", Microsoft Press, 2005, ISBN 0735619174.

To provide background on security requirements, the National Computer Security Center, part of the United States Department of Defense (DoD), published the Trusted Computer System Evaluation Criteria as DoD 5200.28-STD. This has been supplanted by profiles written to the Common Criteria.

The Common Criteria is an ISO standard (ISO/IEC 15408) formal method of specifying requirements for security computer systems. Microsoft Windows uses profiles published by the DoD through the National Information Assurance Partnership (NIAP) program. For more information about NIAP, including the profiles themselves, see "Introducing NIAP" [\[NIAP\]](#) by The National Security Agency.

Microsoft Web sites (see Microsoft Corporation [\[MSFT\]](#)) contain a number of articles about security in Microsoft Windows®. Developer documentation on the MSDN home page [\[MSDN\]](#) by Microsoft also contains more in-depth information.

7 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft Windows NT® operating system

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

[<1> Section 2.2:](#) Supported in Windows NT, Windows 2000, Windows XP, Windows Vista, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

[<2> Section 2.2.1:](#) Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2 domains do not allow the use of a period or multiple periods at the end of a user logon name. The use of a period (.) at the end of a user logon name, as long as the user logon name does not consist solely of period characters, is allowed in the following domains: Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, and Windows NT 4.0.

[<3> Section 2.2.2:](#) Domain join is supported in Windows NT, Windows 2000, Windows XP, Windows Vista, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

[<4> Section 2.2.2:](#) Active Directory is supported in Windows 2000, Windows Server 2003 and Windows Server 2008.

[<5> Section 2.4:](#) Supported in Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

[<6> Section 2.5.2:](#) The list of applicable servers follows: Windows 2000, Windows XP, Windows Vista, Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2.

[<7> Section 2.5.2:](#) Active Directory domain controllers (DCs) that do not accept changes are a new feature for Windows Server 2008 and Windows Server 2008 R2. Windows 2000 servers and Windows Server 2003 servers cannot be configured this way. Windows NT 4.0 domains have one DC that accepts changes, and all other DCs are read-only.

[<8> Section 2.5.3:](#) That is, Netlogon also creates a general purpose channel for authentication. It is not specific to any protocol and is available only to components involved in authentication.

[<9> Section 2.5.5.2:](#) Windows 98, Windows 2000, Windows 2000 Server, Windows Millennium Edition, Windows XP, Windows XP SP1, Windows XP SP2, Windows XP SP3, Windows Vista (RTM version inclusive), Windows Server 2003, Windows Server 2003 with SP1, Windows Server 2003 SP2, Windows Server 2003 with SP3, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2 use algorithms that are compliant with the FIPS 140-2 standard. Windows Vista SP1 uses a new algorithm which is compliant with the NIST SP 800-90 standard. The random number generation in all versions of Windows prior to Windows Vista SP1 is consistent with FIPS 186-2 Change Notice 1.

[<10> Section 2.5.5.2:](#) Windows uses the cryptographic PRNG from the Cryptographic API (CAPI) and the Cryptographic API Next Generation (CNG) for generation of Version 4 GUIDs.

[<11> Section 2.5.5.2:](#) For reasons of increased privacy protection for our customers, Microsoft systems beginning with Windows 2000 prefer to generate version 4 GUIDs in which the 122 bits of nonformat information are random. Although only a small minority of version 4 GUIDs require cryptographic randomness, the random bits for all version 4 GUIDs built in Windows are obtained via the Windows CryptGenRandom cryptographic API or the equivalent, the same source that is used for generation of cryptographic keys. This source is FIPS 140-certified in various versions of Windows, as documented at [\[MSFIP140CryptCerts\]](#)

[<12> Section 3.2.2: \[RFC1510\]](#) (the original Kerberos RFC) was made obsolete and was replaced by [\[RFC4120\]](#).

[<13> Section 5.3:](#) In the following Windows products, the only supported impersonation levels were identify and impersonate: Windows NT 3.1, Windows NT 3.5, Windows NT 3.51, Windows NT 4.0, Windows 98.

Delegate-level impersonation is supported in the following Windows products: Windows 2000, Windows Millennium Edition, Windows XP, Windows XP SP1, Windows XP SP2, Windows XP SP3, Windows Server 2003, Windows Server 2003 with SP1, Windows Server 2003 SP2, Windows Server 2003 with SP3, Windows Server 2003 R2, Windows Vista, Windows Vista SP1, Windows Vista SP2, Windows Server 2008, Windows 7, Windows Server 2008 R2.

[<14> Section 5.3:](#) This level is supported beginning with Windows 2000.

8 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

9 Index

A

Access
 control
 [entry](#) 29
 [lists - discretionary](#) 29
 [rights](#) 31
Accounts
 [computer](#) 8
 domains
 [controllers](#) 14
 [effect](#) 15
 globally unique identifiers (GUIDs)
 [internal structure](#) 16
 [overview](#) 16
 [random bits - quality](#) 17
 [uniqueness](#) 16
 [local](#) 14
 [membership](#) 15
 [overview](#) 14
 [remote](#) 14
 [groups](#) 11
 [overview](#) 7
 [user](#) 8
[ACE](#) 29
Authentication
 [background](#) 19
 [CIFS](#) 27
 [concepts](#) 19
 [history](#) 20
 [Kerberos](#) 23
 [known idiosyncrasies](#) 26
 [model](#) 26
 [NTLM](#) 20
 [overview](#) 19
 [practices](#) 19
 [protocols](#) 20
 [purpose](#) 19
 [SMB](#) 27
 [SPNEGO](#) 25
 [SSL](#) 25
 [TLS](#) 25
 [using](#) 26
Authorization
 [access rights](#) 31
 [ACE](#) 29
 [DACL](#) 29
 [example](#) 31
 [inheritance](#) 32
 [overview](#) 28
 [resource managers](#) 28
 [security descriptors](#) 28
 [system access control list \(SACL\)](#) 32
 [Windows integrity mechanism](#) 33

B

[Bits - random - quality](#) 17

C

[C2](#) 28
[CAPP](#) 28
[Change tracking](#) 42
[CIFS](#) 27
[Cloaking](#) 35
[Computer accounts](#) 8

D

[DACL](#) 29
[DES](#) 20
Domains
 [account](#) 14
 [controllers](#) 14
 [effect on accounts](#) 15
 globally unique identifiers (GUIDs)
 [internal structure](#) 16
 [overview](#) 16
 [random bits - quality](#) 17
 [uniqueness](#) 16
 [local](#) 14
 [membership](#) 15
 [remote](#) 14

E

[Effect - account](#) 15
[Examples - authorization](#) 31

G

Globally unique identifiers (GUIDs)
 [internal structure](#) 16
 [overview](#) 16
 [random bits - quality](#) 17
 [uniqueness](#) 16
Groups
 [nested](#) 13
 [overview](#) 11
 [scope](#) 12
 [types](#) 12

I

Identity
 account domains
 [controllers](#) 14
 [effect](#) 15
 globally unique identifiers (GUIDs)
 [internal structure](#) 16
 [overview](#) 16
 [random bits - quality](#) 17
 [uniqueness](#) 16
 [local](#) 14
 [membership](#) 15
 [overview](#) 14

- [remote](#) 14
- [accounts](#) 7
- [groups](#) 11
- [overview](#) 7
- security
 - [identifiers \(SIDs\)](#) 9
 - [principals](#) 7
- [SIDs](#) 9
- [Idiosyncrasies - authentication](#) 26
- Impersonation
 - [cloaking](#) 35
 - levels
 - [overview](#) 36
 - [setting](#) 37
 - [overview](#) 34
 - [tokens](#) 36
 - [Windows API functions](#) 38
- [Inheritance](#) 32
- [Internal structure - globally unique identifiers \(GUIDs\)](#) 16
- [Introduction](#) 6

K

- Kerberos
 - [description](#) 23
 - [flow](#) 23
 - [history](#) 20
 - [notes](#) 24
 - [overview](#) 23

L

- Levels - impersonation
 - [overview](#) 36
 - [setting](#) 37
- [Local domains](#) 14

M

- [Membership - domain](#) 15

N

- [Nested groups](#) 13
- NTLM
 - [description](#) 20
 - [flow](#) 20
 - [notes](#) 22
 - [overview](#) 20

P

- [Product behavior](#) 40

R

- [Random bits - quality](#) 17
- [References](#) 39
- [Remote domains](#) 14
- [Resource managers](#) 28

S

- [SACL](#) 32
- [Scope of groups](#) 12
- Security
 - [descriptors](#) 28
 - [identifiers \(SIDs\)](#) 9
 - [practices](#) 19
 - [principals](#) 7
 - [tokens](#) 19
- [Setting impersonation level](#) 37
- SIDs ([section 2.3](#) 9, [section 4.2](#) 28)
- [SMB](#) 27
- SPNEGO
 - [description](#) 25
 - [flow](#) 25
 - [history](#) 20
 - [notes](#) 26
 - [overview](#) 25
- [SSL](#) 25
- [System access control lists](#) 32

T

- [TCSEC](#) 28
- [TLS](#) 25
- Tokens
 - [impersonation](#) 36
 - [security](#) 19
- [Tracking changes](#) 42
- [Types of groups](#) 12

U

- [Uniqueness of globally unique identifiers \(GUIDs\)](#) 16
- [User accounts](#) 8

W

- Windows
 - [API impersonation functions](#) 38
 - [integrity mechanism](#) 33
 - [Security Model](#) 39