[MS-AZOD]:
Authorization Protocols Overview

This document provides an overview of the Authorization Protocols Overview Protocol Family. It is intended for use in conjunction with the Microsoft Protocol Technical Documents, publicly available standard specifications, network programming art, and Microsoft Windows distributed systems concepts. It assumes that the reader is either familiar with the aforementioned material or has immediate access to it.

A Protocol System Document does not require the use of Microsoft programming tools or programming environments in order to implement the Protocols in the System. Developers who have access to Microsoft programming tools and environments are free to take advantage of them.

Abstract

This document provides an overview of the functionality and relationship of the Authorization protocols, which control the process of granting access to resources once authentication has been accomplished. An authenticated request is not sufficient for access by itself; a corresponding decision must also be made to decide if a particular request is authorized. To accomplish this, several authorization models are provided under Windows. This document provides an overview of these models as implemented by [MS-PAC], [MS-AZMP], [MS-GPCAP], [MS-CAPR], [MS-CTA], [MS-DTYP], [MS-ADTS], [MS-COMA], and [MS-TDS].

This document describes the intended functionality of the Authorization Protocols and how these protocols interact with each other. It provides examples of some common use cases. It does not restate the processing rules and other details that are specific for each protocol. Those details are described in the protocol specifications for each of the protocols and data structures that belong to this protocols group.

Revision Summary

Date	Revision History	Revision Class	Comments
10/25/2012	1.0	New	Released new document.
01/31/2013	1.0	No change	No changes to the meaning, language, or formatting of the technical content.
08/08/2013	1.1	Minor	Clarified the meaning of the technical content.

Contents

1 Introduction

1.1 Conceptual Overview

Authorization is the process of controlling access to resources. Once authentication has been accomplished, the next task is to decide if a particular request is authorized. Management of network systems often models broad authorization decisions through roles, groups and claims; for example, all engineers who have access to a specific printer, all sales personnel who have access to a certain web server, or confidential information where access is allowed only to certain authorized user groups or users based on the **claims** configured. Making authorization information consistently available to a number of services allows for simpler management.

The authorization system always deals with two entities, the security principal (subject) and the resource (object) or business operation/task, as depicted in the following figure. When a **security principal** wants to access a resource or perform a business operation/task, the authorization system checks all accesses requested by the security principal.

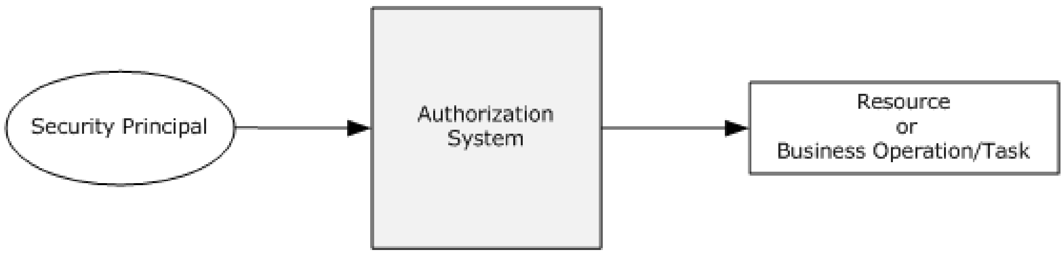

Figure 1: Generic authorization model

To perform the tasks that they are designed for, applications must carry out operations and access system resources on behalf of the application's user while protecting these operations and resources from unauthorized access. Administrators can control whether a process can access securable objects or perform various system administration tasks.

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 specified in [CCITSE3.1-3], such as the Controlled Access Protection Profile.

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

This control was initially provided through the Discretionary Access Control (DAC) Model, which is an object-centric model using **access control lists (ACLs)**. Each system object has an associated list of trustees (user account, group account) with specific sets of access rights for that object. This model lends itself well to securing access to well-defined, persistent resources such as **Active Directory**, file, and registry.

Windows Server 2003 operating system introduced a complementary authorization interface, called Authorization Manager (AzMan), which enables the **role-based access control (RBAC)** authorization model. Authorization Manager provides a natural framework for business process applications that require representing the organizational model within the application security framework.

In the DAC model, a resource manager (RM) manages its own set of objects, which are protected by a **security descriptor**. Whenever a client requests access to a resource protected by an RM, the RM makes a call to the authorization system to verify the authorization of the client's identity. In turn, the authorization system looks at the client security token, the desired access to the object, and the security descriptor on the object. The authorization system returns to the RM, responding "yes" or "no," providing the RM the ability to determine whether the client should be allowed to access the object.

In contrast to object-centric management, AzMan RBAC provides a framework for developers to develop applications that are oriented around the notion of the role. Rather than managing access control on objects in the application, AzMan RBAC facilitates application development by providing a central object—a role—that a user is assigned to perform a particular job function within an application. A role directly implies authorization permissions on some defined set of resources.

Through the abstractions of the operation and task, AzMan RBAC permissions are typically granted through higher level abstractions corresponding to high-level tasks defined by the application developer. Operations represent a single unit of application code, whereas tasks may be composed of multiple operations (and other tasks). Consider an example, a web-based application that allows users to report project status, publish status for viewing, and view status. The COM development framework also has the notion of an application-specific role; this role is very similar to the one used in the context of the AzMan RBAC model, and the key difference with the AzMan RBAC is that the COM+ roles access control model can only be used in COM/COM+ applications, whereas the AzMan RBAC model can be integrated into any application type.

This section provides an overview of the following concepts, which are required for understanding the document.

1.1.1 DAC Model

1.1.1.1 Authorization Information (PAC)

For a server implementation of an authentication protocol, the result of the authentication produces a variety of data. Some of the data is related to the authentication protocol, such as keys for encrypted communication, and is covered in the relevant authentication protocol specification. Additionally, after the identity of the client is determined, additional data corresponding to authorization of the client to the server is derived. This authorization information is frequently referred to as Privilege Attribute Certificate (PAC), and it contains group memberships and claims, or group memberships from the domain controller. Each authentication protocol uses its own specific data structure to carry the authorization information. This table lists the mapping of authentication protocol with authorization structures.

Authentication protocol	Authorization data structure	Reference technical documents
Kerberos Protocol Extensions	Privilege attribute certificate	[MS-PAC]
Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol	Privilege Attribute certificate	[MS-PAC]
NT LAN Manager (NTLM) Authentication Protocol	NETLOGON_VALIDATION_SAM_INFO	[MS-APDS]

Authentication protocol	Authorization data structure	Reference technical documents
		[MS-NRPC]
Digest Protocol Extensions	Privilege Attribute certificate	[MS-PAC] [MS-APDS]
Secure Sockets Layer (SSL)/Transport Layer Security (TLS) Protocols	Privilege Attribute certificate	[MS-PAC] [MS-RCMP]

1.1.1.2 Security Identifiers (SIDs)

The SID, as specified in [MS-DTYP] section 2.4.2, is an account identifier. It 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)**. An example of a two-subauthority **SID** is shown in the following figure.

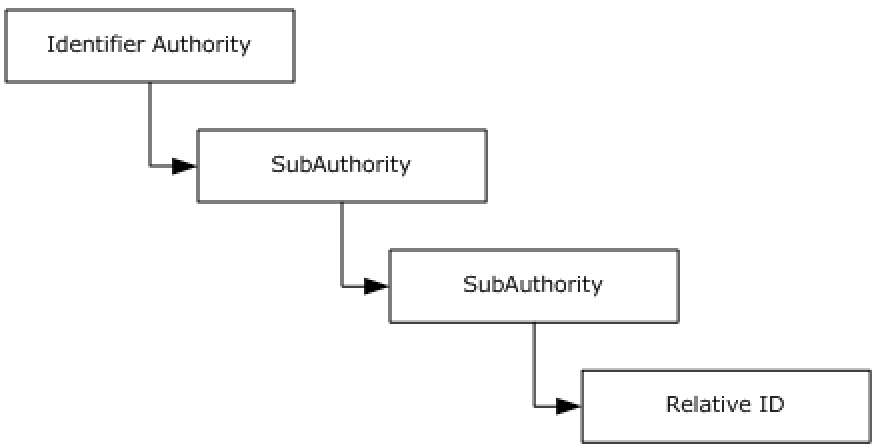

Figure 2: Windows SID with subauthorities

The original definition of a 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 Windows development; however, the structure was 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. When the new issuing authority for which this SID was created is a domain, this SID is known as a "domain SID".

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.

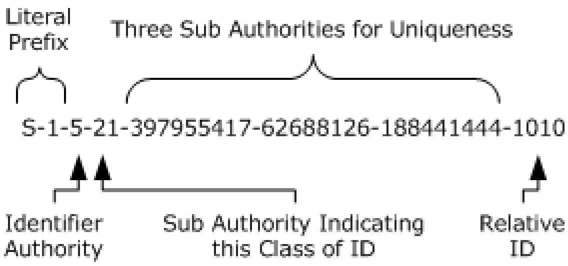

Figure 3: SID with account association

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 simple approach to this entails allocating RIDs sequentially. More complicated schemes are certainly possible. For example, 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 be sure to handle this situation correctly. Typically, the 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 between a SID that is assigned to a group or one assigned to an account. Changing the name of a user, computer, or domain does not change the underlying SID for an account. Administrators cannot modify the SID for an account, 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.

1.1.1.3 Security Descriptor

The security descriptor is the basis for specifying the security associated with an object. Every object that has a security descriptor linked to its object is called a securable object. Securable objects can be shared between different users--and every user can have different authorization settings. Examples of securable objects are a file, a folder, a file system share, a printer, a registry key, and an Active Directory object. The following figure depicts the abstract representation of the security descriptor data structure.

The security descriptor is a collection of four main elements, as shown in the following figure: the owner, the group, the DACL, and the SACL.

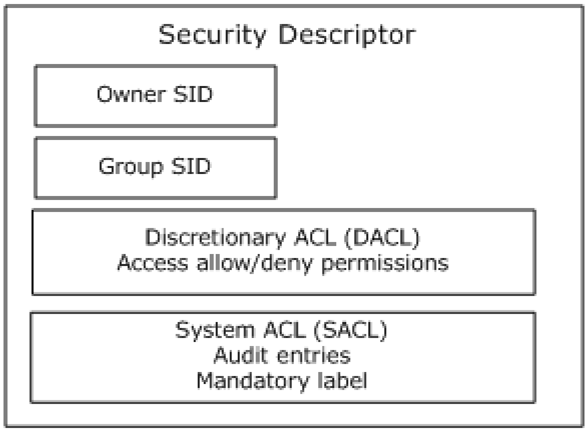

Figure 4: Abstract representation of security descriptor

The **Owner** is a SID that specifies the owner of the resource. The **Group** SID specifies the group associated with the resource. The Group SID field is not evaluated by Windows components, and it exists for Portable Operating System Interface for UNIX (POSIX)compatibility. The **DACL** field specifies the **discretionary access control list**, and the **SACL** field specifies the **system access control list**.

When associated with a resource, the security descriptor is intended to be opaque. The resource manager (RM) should never be required to examine the contents of the security descriptor. However, the security descriptor fields can be used by the RM for other purposes. For example, in a billing scenario, the file system can implement a storage quota system by using the owner field in the security descriptor to determine the resources consumed with a specific user. Security Descriptor algorithms are defined in [MS-DTYP] section 2.5.3.

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.

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. [MS-DTYP] section 2.4.6 describes the SECURITY_DESCRIPTOR structure.

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 protocols check the object's DACL, looking for ACEs that apply to the user and group SIDs referenced in the user's access token. The security protocols then step 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 protocols do 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 evaluation process for an access token and a DACL when a request is evaluated.

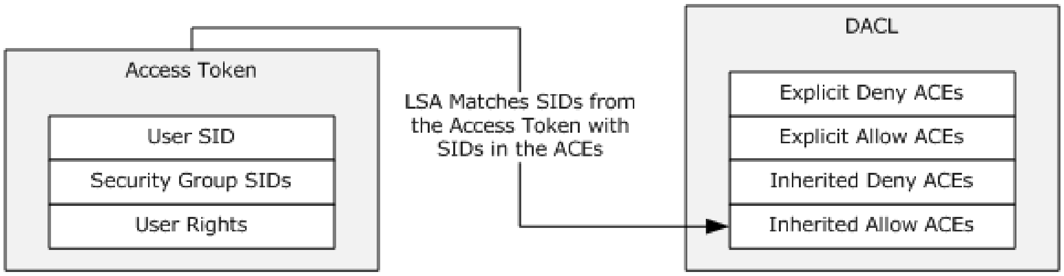

Figure 5: Evaluation process for access tokens against a DACL

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: that is, anyone with ownership of a file or folder, Change permissions, or Full Control permissions can assign access control at their discretion. When Windows is first installed, Windows assigns default permission structures to operating system files and folders when it is newly installed, but a user might be required 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, which means 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.

A system access control list (SACL) enables administrators to log attempts to access a secured object. Like a DACL, a SACL is a list of ACEs. Each ACE specifies the types of access attempts made by a specified account, which 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. For more details about the security descriptor see [MS-DTYP] section 2.4.6.

The security descriptor of a file system object is stored in the NTFS file system, whereas the security descriptor of an Active Directory object is stored in the object's **nTSecurityDescriptor** (see [MS-ADA3] section 2.37) attribute. See [MS-DTYP] section 2.5.3.4, Algorithm for Creating a Security Descriptor.

The SECURITY_DESCRIPTOR structure ([MS-DTYP] section 2.4.6) is a compact binary representation of the security associated with an object in a directory or on a file system, or in other stores. However, it is not convenient for use in tools that operate primarily on text strings. Therefore, a text-based form of the security descriptor is available for situations when a security descriptor must be carried by a text method. This format is the Security Descriptor Description Language (SDDL). For more information on this, see [MS-DTYP] section 2.5.1.

1.1.1.4 Resource Managers

In the DAC model, a **resource manager (RM)** is the code or component that implements one or more securable object types. Many RMs--including the file system, registry, Active Directory, and operating system constructs, such as processes--exist in a Windows-based system. The NTFS file system is a resource manager that implements files and directories; the Windows registry is a resource manager that implements keys. Even though these RMs control very different objects, they share a common method for controlling access.

Windows also distinguishes between ordinary objects in the RM and containers exposed by the RM. In the file system, files are objects and directories are containers. This distinction is important during the creation of new objects.

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 be able to retrieve the security descriptor for an object when authorization validation is required and is not required to understand the contents.

1.1.1.5 Access Rights

The access mask or rights tells the authorization system what the process (which is acting on user's identity) wants to do with a resource, for example, read a file or write to a file. For more details, see [MS-DTYP] section 2.4.3.

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.

Windows accomplishes this by partitioning the access rights space. Access rights can be 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 enumerate subkeys and bit 2 to read a key's value.

Additionally, DAC supports defining access rights using GUIDs, and in this way arbitrary number of access rights can be defined. Active Directory uses this model as described in [MS-ADTS] section 5.1.3.2.1 and section 5.1.3.2.2.

The following table lists out the mapping of resource managers with the corresponding access rights data structure.

Resource manager type	Access rights reference
Active Directory objects	Section 5.1.3.2 in [MS-ADTS]
NTFS objects	Section 2.2.13.1 in [MS-SMB2] Section 2.2.1.4 in [MS-SMB]
Registry objects	Section 2.2.4 in [MS-RRP]
Printer objects	Section 2.3.1 in [MS-RPRN] Section 3.1.1.4.1 in [MS-PAN]

1.1.1.6 User Rights

User rights are the authority to perform an operation that affects an entire computer rather than a particular object. User rights are assigned by administrators to individual users or groups as part of the security settings for the computer. Although user rights can be managed centrally through Group Policy, they are applied locally. Users can (and usually do) have different user rights on different computers.

User rights can be split into two categories: logon rights and user privileges. Logon rights control who can log on to a computer system and how he or she can do the logon. User privileges are used to control access to system resources and system-related operations, such as changing the system time or the ability to shut down the system.

User rights grant specific privileges and logon rights to users and groups in a computing environment.

For a list of privileges that are supported in Windows versions, see [MS-LSAD] section 3.1.1.2.1, and for logon rights, see [MS-LSAD] section 3.1.1.2.2.

1.1.1.7 Access Token

Authorization contexts are built from the authorization information that is obtained during or after the authentication process, from server-local information, or a combination of the two, depending on implementation choices.

The authorization context is also referred to as the access token, which is a collection of the groups and claims associated with the client principal and potentially the device(such as a computer) from which the client is connecting, as well as additional optional policy information. The authorization context plays a central role in determining access, through the evaluation of a security descriptor. Note that the token is never passed directly across the network; tokens are local information and the actual representation is up to the implementation. This token is represented as an abstract data structure as shown in the following figure.

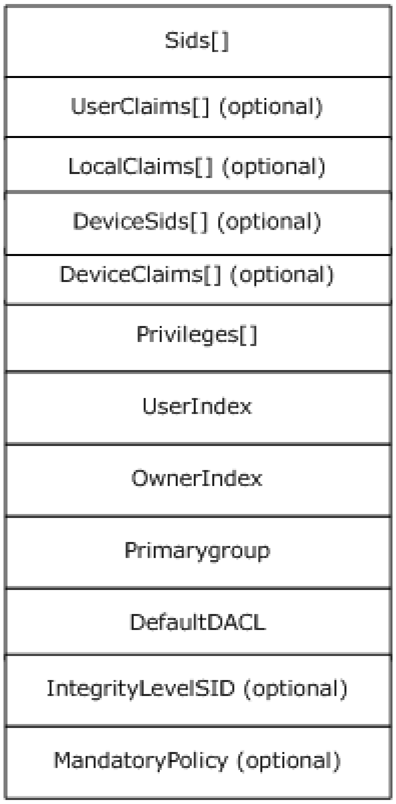

Figure 6: Access Token abstract representation

For descriptions of Access Token structure fields, refer to section 2.5.2 in [MS-DTYP], and for more information about tokens in Windows, see [MSDN-ACCTOKENS].

1.1.1.8 Impersonation

In distributed systems, it is typical for a server to accomplish tasks on behalf of a client. The functionality of a server performing a task using the security context of a client to access the server's local resources is called impersonation.

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.

See section 2.7 in [MS-DTYP] for Impersonation Abstract Interfaces. Additional references for information on Windows impersonation include the following:

- Delegation and Impersonation: [MSFT-DAI]
- Client Impersonation (RPC): [MSFT-RPCCI]
- Client Impersonation (API functions): [MSDN-CI]

1.1.1.9 Inheritance

The DAC 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 DAC), 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. A **Container-Inherit** flag 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. An **Inherit-Only** flag indicates that when a child object is created, this ACE should be carried forward to the security of the child object if either an **Object-Inherit** or a **Container-Inherit** flag is present on the parent (container) object. This inherit-only ACE does not control access to the object to which it is attached. See [MS-DTYP] section 2.4.4.1 for more details.

1.1.1.10 Windows Integrity Mechanism

Beginning with Windows Vista operating system, the 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 (see [MS-DTYP] section 2.4.6). Windows restricts access rights depending on whether the subject's integrity level is equal to, higher than, or lower than the object's integrity level. The integrity level of an object is stored as a mandatory label ACE that distinguishes it from the discretionary ACEs governing access to the object.

The ACE represents the object integrity level. An integrity level is also assigned to the 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 authorization system performs an access check. See [MS-DTYP] section 2.5.3.3 for an example of the MandatoryIntegrityCheck Algorithm pseudo-code. The security subsystem implements the integrity level as a mandatory label to distinguish it from the discretionary access (under user control) that DACLs provide. See [MSDN-WIMD] for a discussion of Windows integrity mechanism design.

1.1.1.11 Claim Based Access Control (CBAC) Model

Conditional ACEs or expressions were introduced to the authorization system to allow it to make its access control decisions not only based on the identity of the trustees, but also whether trustees met the particular conditions. A user access request can be granted or denied by comparing the ACLs on the security descriptor with the attributes (called claims) of the user access token. For more details on the conditional ACEs, see section 2.4.4.17 in [MS-DTYP].

A claim is an attribute that makes an assertion about an entity with which it is associated. Claims are broadly classified in three categories based on entity: user claims, device claims, and resource properties or claims.

User claim: A claim that is associated with an authenticated user account. Examples of user claims are employer of the user, type of the employment, role in organization, and organization division of the user.

Device claim: A claim that is associated with an authenticated computer account. Along with the claims, it can be included in the user token of the user trying to access the resource. Examples of device claims are the IT management status of the computer and the department the computer is designated to operate in.

Resource property: A property that is associated with the resource on the system. Examples of resource properties are classification of the resource such as High-Business-Impact, Confidential, and Personally-Identifiable-Information..

CBAC is an access control paradigm that utilizes the claims to make access-control decisions to resources. In Windows, CBAC is built upon on the conditional ACEs feature, not to just utilize the user claims, but also to utilize the resource claims (referred to as resource properties) in order to make access-control decisions. If the resource also has a resource claim "Division" that is equal to Sales, the policy condition can be stated using the SDDL syntax

"O:BAG:BAD:(XA; ;FX;;;S-1-1-0;(@User. Division==@Resource. Division))"

Using this approach, the "Division" claim of the resource can be separately defined and changed without needing to update the conditional expression on the resource.

1.1.2 AzMan RBAC Model

1.1.2.1 Roles, Tasks, and Operations

In contrast to the DAC model, which is oriented around objects, the AzMan RBAC model attempts to orient the common administrative experience around user roles. Rather than assigning permissions to objects, an AzMan RBAC framework enables applications to present administrators with a management experience more aligned with the organizational structure of a company. AzMan RBAC provides a central object--a role--that a user is assigned to perform a particular job or application function. Ideally, an RBAC application is designed such that the administrator requires less knowledge of the object storage structure. This can be done if the RBAC application provides a simplifying abstraction into resource collections referred to as scopes. A role directly implies authorization permissions on some scope of resources, as illustrated in the following figure.

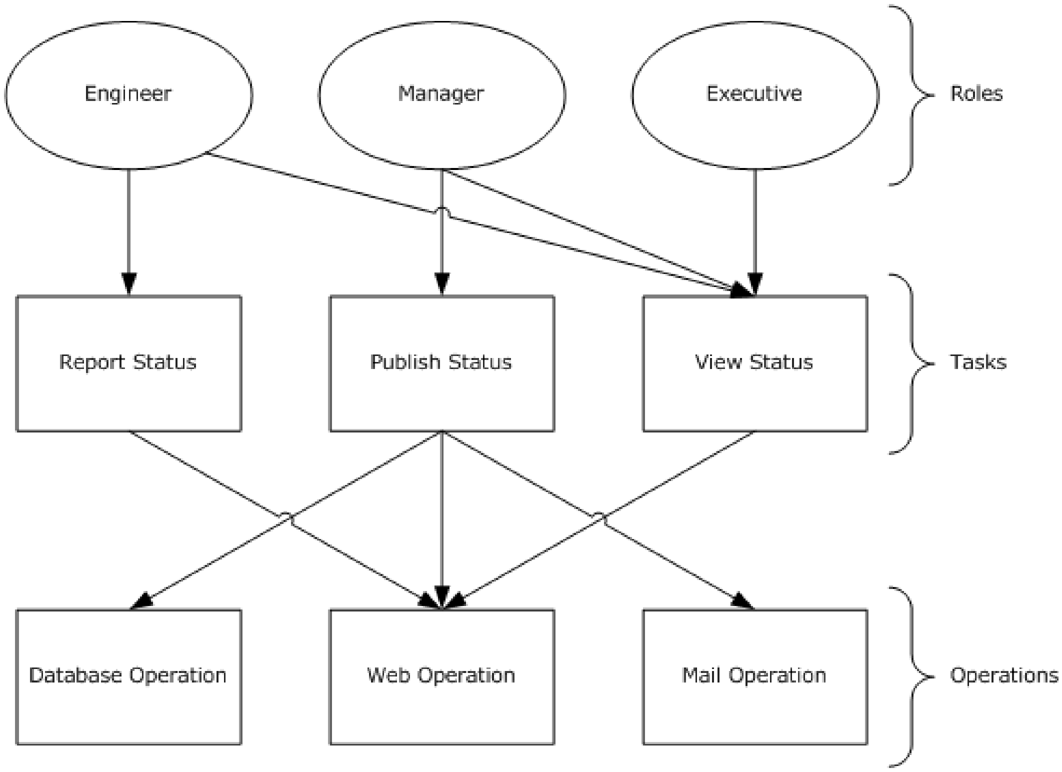

Figure 7: AzMan RBAC permissions access work flow

In the AzMan RBAC model, the role is the interface an administrator uses to manage permissions and assignments. For example, a company can create a role called "Engineer" that is defined in terms of the permissions engineers need for their jobs. Each engineer hired is assigned to the "Engineer" role and instantly has all required permissions for that job. Similarly, engineers who leave the position of engineer are removed from the "Engineer" role and no longer have engineer access. Whereas ACLs work well for well-defined, persistent resources, the role-based model lends itself well to protecting workflow or groups of multiple distinct operations (for example, "read from database" and "send email") to be performed by the application. The preceding figure illustrates the "Engineer" role with permission to report and view status, the "Manager" role with permission to publish and view status, and the "Executive" role with permission to view status.

In Windows, the Authorization Manager Framework provides an interface for developing RBAC applications.

1.1.2.2 Application-Scoped Groups

AzMan RBAC also allows users to be collected into groups. AzMan RBAC groups are similar to groups in the Active Directory service, but they are maintained for a specific set of applications, a single application, or a scope within an application.

Authorization Manager introduces three types of application-scoped groups:

- **Application Basic Group:** Similar to Windows security groups, the application basic group contains a list of members. Unlike Windows security groups, it also has an additional list for nonmembers. The nonmembers list allows for exceptions, so a large group can be used but a smaller group or particular user can be excluded.

- **Lightweight Directory Access Protocol Query Group:** A group defined by an LDAP query (see [RFC4511]) against the attributes of a given Active Directory user's account. At the time of access, the LDAP query is run to determine if the user is a member of that group. This allows for flexible group membership that remains up-to-date with the user's Active Directory account object. For example, a Managers group could contain an LDAP query that includes all users who have direct reports.
- **BizRule-based group**: This group allows membership to a group to be based on the AzMan BizRule script evaluation.

1.1.2.3 Authorization Store

The object-based authorization framework maintains access rights in DACLs on the objects. In the role-based model however, security information is maintained in a separate location from objects, in a policy store.

In Windows, the Authorization Manager allows authorization policy to be stored in either Active Directory or in files in .xml format or on an SQL server. Administrators on the system that contains the authorization policy store have a high degree of access to the store, so the authorization policy store must be located on a trusted system.

When using the Active Directory store, Authorization Manager creates Active Directory objects for the store itself and child objects for each application group, application, operation, task, role, and scope. The scope object can contain tasks, roles, and groups created in that scope.

Authorization Manager also allows the authorization policy to be stored in .xml format on a file stored on an NTFS file system (protected by an ACL). The XML store can be kept on the same computer as an Authorization Manager server or it can be stored remotely.

1.1.3 COM + Roles Access Control Model

See [MSDN-COM+ Security] for details on the COM+ roles access control model.

1.2 Glossary

The following terms are defined in [MS-GLOS]:

access control entry (ACE)
access control list (ACL)
Active Directory
claim
discretionary access control list (DACL)
domain controller (DC)
forest (1)
Group Policy server
globally unique identifier (GUID)
Key Distribution Center (KDC)
KRB_AP_REQ/KRB_AP_REP
Local Security Authority (LSA) database
privilege (1)
privilege attribute certificate (PAC)
security account manager (SAM) built-in database
security descriptor
security principal (2)
security identifier (SID)

service ticket
ticket-granting ticket (TGT)
system access control list (SACL)

The following terms are defined in [MS-KILE]:

Flexible Authentication Secure Tunneling (FAST)
integrity level

The following terms are defined in [MS-GPCAP]:

central access policy (CAP)
central access rule (CAR)

The following terms are specific to this document:

Active Directory client (AD client): The **AD client** is the application running on the client computer, and the user who is the primary actor uses this application to access objects or attributes of the **Active Directory**. The **AD client** application uses the **Active Directory** protocols as described in [MS-ADOD].

Active Directory server (AD server): The **AD server** is the service or process running on the server computer under the security context of the identity of the **AD client**.

application client: The **application client** is the application running on the client computer and the user who is the primary actor uses this application to perform required business operation/tasks.

claim definition: A definition that indicates the identifying name of the **claim** and the types of the values to which this attribute can be assigned.

compound identity: The combination of user and computer account identities.

file client: The **file client** is the application which implements client-side of the file access protocol components and that enables the primary actor (user) to access the shared files on remote file server.

file server: The **file server** is the service or process on a server computer, which implements the server-side file access protocol components to enable remote file sharing for the **file clients**.

role-based access control (RBAC): The authorization-manager-based access-control paradigm that controls the access to the resources or business process based on the role permissions.

1.3 References

[MS-ADA3] Microsoft Corporation, "Active Directory Schema Attributes N-Z".

[MS-ADOD] Microsoft Corporation, "Active Directory Protocols Overview".

[MS-ADSC] Microsoft Corporation, "Active Directory Schema Classes".

[MS-ADTS] Microsoft Corporation, "Active Directory Technical Specification".

[MS-APDS] Microsoft Corporation, "Authentication Protocol Domain Support".

[MS-AUTHSOD] Microsoft Corporation, "Authentication Services Protocols Overview".

[MS-AZMP] Microsoft Corporation, "Authorization Manager (AzMan) Policy File Format".

[MS-CAPR] Microsoft Corporation, "Central Access Policy Identifier (ID) Retrieval Protocol".

[MS-CIFS] Microsoft Corporation, "Common Internet File System (CIFS) Protocol".

[MS-COMA] Microsoft Corporation, "Component Object Model Plus (COM+) Remote Administration Protocol".

[MS-CTA] Microsoft Corporation, "Claims Transformation Algorithm".

[MS-DPSP] Microsoft Corporation, "Digest Protocol Extensions".

[MS-DTYP] Microsoft Corporation, "Windows Data Types".

[MS-FASOD] Microsoft Corporation, "File Access Services Protocols Overview".

[MS-FCIADS] Microsoft Corporation, "File Classification Infrastructure Alternate Data Stream (ADS) File Format".

[MS-FSA] Microsoft Corporation, "File System Algorithms".

[MS-FSMOD] Microsoft Corporation, "File Services Management Protocols Overview".

[MS-FSRM] Microsoft Corporation, "File Server Resource Manager Protocol".

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-GPCAP] Microsoft Corporation, "Group Policy: Central Access Policies Protocol Extension".

[MS-KILE] Microsoft Corporation, "Kerberos Protocol Extensions".

[MS-LSAD] Microsoft Corporation, "Local Security Authority (Domain Policy) Remote Protocol".

[MS-NLMP] Microsoft Corporation, "NT LAN Manager (NTLM) Authentication Protocol".

[MS-NRPC] Microsoft Corporation, "Netlogon Remote Protocol".

[MS-PAC] Microsoft Corporation, "Privilege Attribute Certificate Data Structure".

[MS-PAN] Microsoft Corporation, "Print System Asynchronous Notification Protocol".

[MS-PKCA] Microsoft Corporation, "Public Key Cryptography for Initial Authentication (PKINIT) in Kerberos Protocol".

[MS-PRSOD] Microsoft Corporation, "Print Services Protocols Overview".

[MS-RAA] Microsoft Corporation, "Remote Authorization API Protocol".

[MS-RCMP] Microsoft Corporation, "Remote Certificate Mapping Protocol".

[MS-RDSOD] Microsoft Corporation, "Remote Desktop Services Protocols Overview".

[MS-RPRN] Microsoft Corporation, "Print System Remote Protocol".

[MS-RRP] Microsoft Corporation, "Windows Remote Registry Protocol".

[MS-SFU] Microsoft Corporation, "Kerberos Protocol Extensions: Service for User and Constrained Delegation Protocol".

[MS-SMB] Microsoft Corporation, "Server Message Block (SMB) Protocol".

[MS-SMB2] Microsoft Corporation, "Server Message Block (SMB) Protocol Versions 2 and 3".

[MS-SPNG] Microsoft Corporation, "Simple and Protected GSS-API Negotiation Mechanism (SPNEGO) Extension".

[MS-TDS] Microsoft Corporation,"Tabular Data Stream Protocol".

[MS-TLSP] Microsoft Corporation, "Transport Layer Security (TLS) Profile".

[MSDN-ACCTOKENS] Microsoft Corporation, "Access Tokens", http://msdn.microsoft.com/en-us/library/aa374909.aspx

[MSDN-AuthMgr] Microsoft Corporation, "Developing Applications Using Windows Authorization Manager", http://msdn.microsoft.com/en-us/library/aa480244.aspx

[MSDN-CI] Microsoft Corporation, "Client Impersonation", http://msdn.microsoft.com/en-us/library/aa376391.aspx

[MSDN-COM+ Security] Microsoft Corporation, "COM+ Security", http://msdn.microsoft.com/en-us/library/windows/desktop/ms681314(v=vs.85).aspx

[MSDN-WIMD] Microsoft Corporation, "Windows Integrity Mechanism Design", http://msdn.microsoft.com/en-us/library/bb625963.aspx

[MSFT-RPCCI] Microsoft Corporation, "Client Impersonation (RPC)", http://msdn.microsoft.com/en-us/library/windows/desktop/aa373582(v=vs.85).aspx

[RFC4511] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006, http://www.rfc-editor.org/rfc/rfc4511.txt

2 Functional Architecture

2.1 Overview

This section details the overviews of DAC, RBAC and COM+ roles authorization models.

2.1.1 System Capabilities

The Authorization protocols enable the applications to make access control decisions. In Windows, the authorization system has the capability to support the following authorization models:

- DAC and CBAC models
- AzMan RBAC model
- COM+ roles access control model

The following table illustrates the features of the DAC model that are implemented in Windows resource managers.

Authorization feature	Active Directory objects	NTFS objects	Registry objects	Printer objects
Inheritance (see [MS-DTYP] section 2.5.3.4)	Yes	Yes	Yes	Yes
Object-specific access (see [MS-ADTS] section 5.1.3.3.3)	Yes	No	No	No
Control access rights (see [MS-ADTS] section 5.1.3.2.1)	Yes	No	No	No
Validated write rights (see [MS-ADTS] section 5.1.3.2.2)	Yes	No	No	No
Object visibility	Yes	No	No	No
Conditional expression ACEs	No	Yes	No	No
Claims (CBAC)	No	Yes	No	No

2.1.2 Applicability

The DAC model is suited for the well-defined persistent resources such as Active Directory, files, and the registry. CBAC is an extension to the DAC model, applicable for file resources on a file server.

The Authorization Manager-based RBAC model provides a natural framework for business process applications that require representing the organizational model within the application security framework. In Windows, remote desktop gateway applications use this model.

The COM+ roles authorization model is applicable for the applications that are developed using COM and COM+ development frameworks.

2.1.3 Authorization Process

Windows determines access so that the results are always predictable and consistent. The authorization process is as follows:

To determine access, the calling RM supplies the security descriptor (which contains the ACL) with the identity of the user and all of the groups of which the user is a member, as well as the access requested by the user. The following example can be used to illustrate the authorization process:

```
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, and therefore 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 process is 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.

2.1.4 DAC Model

2.1.4.1 Protocol Communications

2.1.4.1.1 Kerberos Protocol Extensions

The following figure illustrates the protocol interactions when using Kerberos Protocol Extensions (KILE) (see [MS-KILE]) or Public Key Cryptography for Initial Authentication (PKCA) (see [MS-PKCA]) as the authentication protocol.

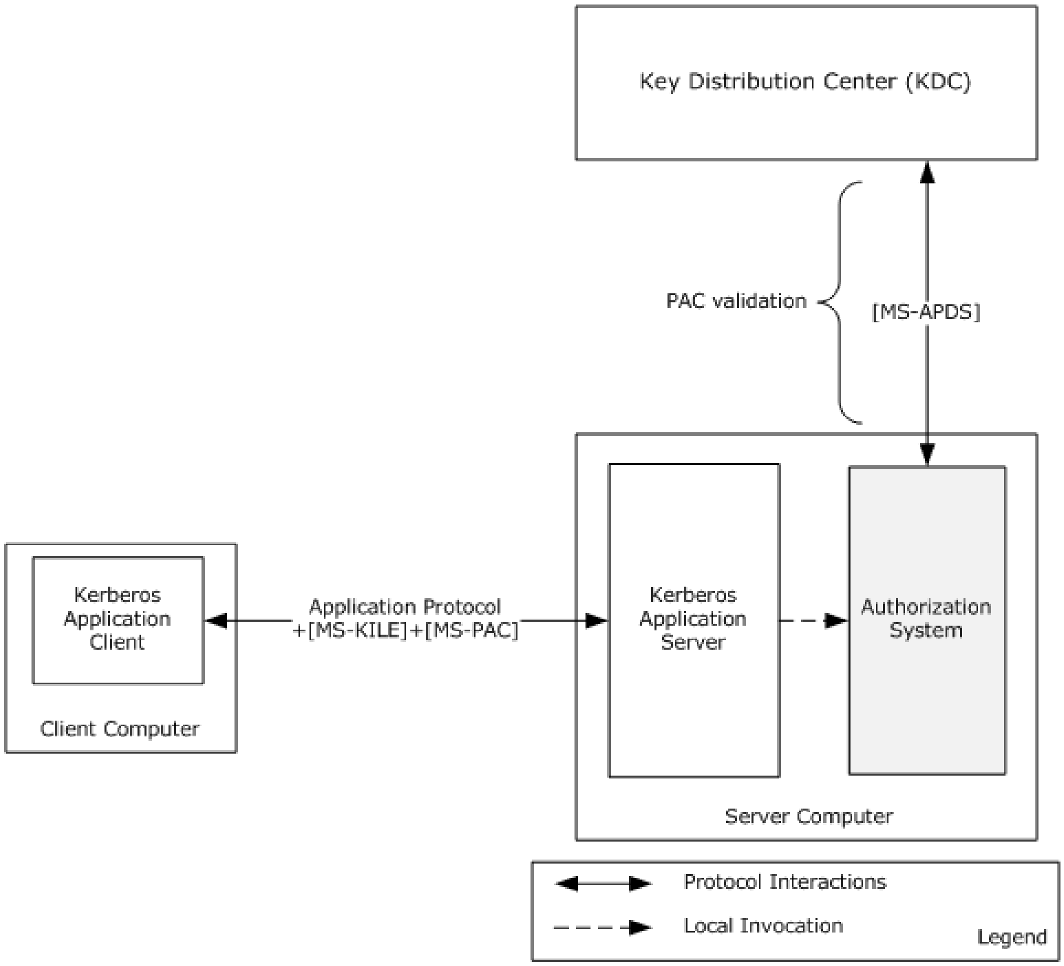

Figure 8: Protocol interactions when the authentication protocol is KILE or PKCA

The identity of the Kerberos application client has been authenticated using either the KILE or PKCA protocol and obtained the service ticket for the Kerberos application server as described in [MS-AUTHSOD] section 2.1.2.3. The Kerberos application client submits the service ticket (with the user's authorization information, as described in [MS-PAC]) in a *KRB_AP_REQ* message to the Kerberos application server using an application-specific protocol.

The Kerberos application server validates the received *KRB_AP_REQ* message to verify the identity of the requesting user, and if the verification succeeds, then the Kerberos application server validates the Server Signature ([MS-PAC] section 2.8.1) in the Privilege Access Certificate (PAC) as described in [MS-PAC]. If tampering with the PAC could result in inappropriate elevation of privileges, then in addition to validating the server signature, the **Key Distribution Center (KDC)** signature will be validated. If PAC validation is required (see [MS-APDS] for the requirements of PAC validation), then the authorization system forwards the PAC signature in the *KRB_AP_REQ* message to the **domain controller** for verification in a *KERB_VERIFY_PAC* message, as described in [MS-APDS] section 3.2, or else it will directly proceed to constructing the access token. The authorization system constructs the access token with the group membership information from PAC, local security groups from SAM database, and privileges and logon rights from the **LSA policy database**.

The application server impersonates the user using this access token, and invokes the access check function in authorization system (through resource manager) by passing the access token, access mask, and security descriptor of the requested object. The authorization system executes the access check algorithm as described in [MS-DTYP] section 2.5.3.2 to verify whether the requested identity has sufficient access permissions to access the object.

2.1.4.1.2 NT LAN Manager (NTLM) Authentication Protocol

The identity of the application client has been authenticated using the NT LAN Manager Authentication Protocol Specification (NTLM) and Authentication Protocol Domain Support Specification (APDS) protocols, as described in [MS-AUTHSOD] section 2.1.2.3. After the authentication process succeeds, the domain controller will return a **NETLOGON_VALIDATION_SAM_INFO*** structure. The authorization system builds the access token with the group membership information from the **NETLOGON_VALIDATION_SAM_INFO*** structure, local security groups from the SAM database, privileges, and logon rights from the LSA policy database.

The application server impersonates the identity access token, and invokes the access check function in the authorization system by passing the access token, access mask, and security descriptor of the requested object. The authorization system executes the access check algorithm as described in [MS-DTYP] section 2.5.3.2 to verify whether the requested identity has sufficient access permissions to access the object.

2.1.4.1.3 Digest Protocol Extensions

The identity of the application client has been authenticated using the MS-DPSP and [MS-APDS] protocols as described in [MS-AUTHSOD] section 2.1.2.4. After authentication, the DC creates and sends back the *DIGEST_VALIDATION_RESP* message (see section 2.2.3.2 in [MS-APDS]) with authorization information for the user's account (the PAC).

The next step of the application server is to verify the access permissions of the user. The application server contacts the authorization system to get the access token by submitting the user's authorization information received from the DC. The authorization system builds the access token with the user's authorization information, local security groups from the SAM database, and privileges and logon rights from the LSA policy database, and returns the access token to the application server.

The application server impersonates the user with the user's access token, and invokes the access check function in the authorization system (through object’s resource manager) by passing the access token, access mask, and security descriptor of the requested object. The authorization system executes the access check algorithm as described in [MS-DTYP] section 2.5.3.2 to verify whether the requested identity has sufficient access permissions to access the requesting object.

2.1.4.1.4 SSL/TLS Protocol

The identity of the application client has been authenticated using the SSL/TLS (see [MS-TLSP]) and RCMP (see [MS-RCMP]) protocols as described in [MS-AUTHSOD] section 2.1.2.4.

On a successful authentication, the domain controller generates the *SSL_CERT_LOGON_RESP* message, which includes the user's PAC, as specified in [MS-PAC], and sends the message back via the Netlogon Remote Protocol ([MS-NRPC]). On receiving this message, the server generates an access token.

The application server impersonates the user using this access token, and invokes the access check function in the authorization system (through the resource manager) by passing the access token, access mask, and security descriptor of the requested object. The authorization system executes the access check algorithm as described in [MS-DTYP] section 2.5.3.2 to verify whether the requested identity has sufficient access permissions to access the object.

2.1.4.2 Internal Components

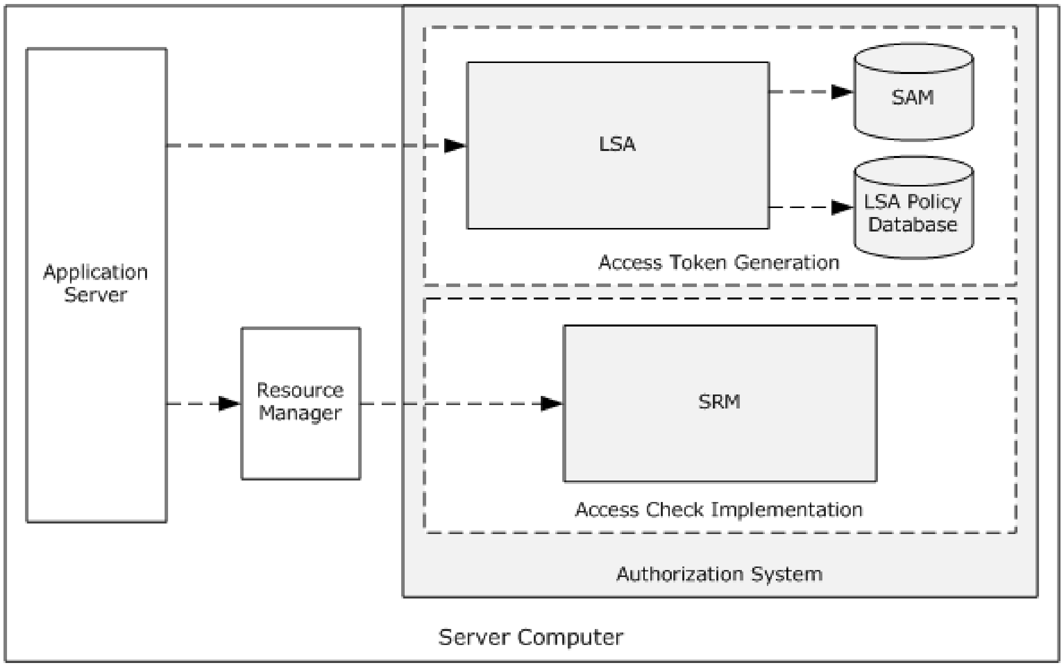

Figure 9: Internal components of the DAC system

The preceding diagram shows the internal components of the DAC system.

The **Local Security Authority (LSA)** is the security subsystem in Windows. This component is responsible for creating the access token with the user authorization information (PAC), privileges from the LSA policy database, and local security groups from the Security Account Manager (SAM) database.

The **Security Reference Monitor (SRM)** is the component of Windows that implements the authorization system. It is the only security component of Windows that is running in the highly privileged OS Kernel mode. It implements the access check algorithm and it checks access to resources by comparing the access control entries in the security descriptor with the group membership information in the user's access token.

2.1.4.3 CBAC Model

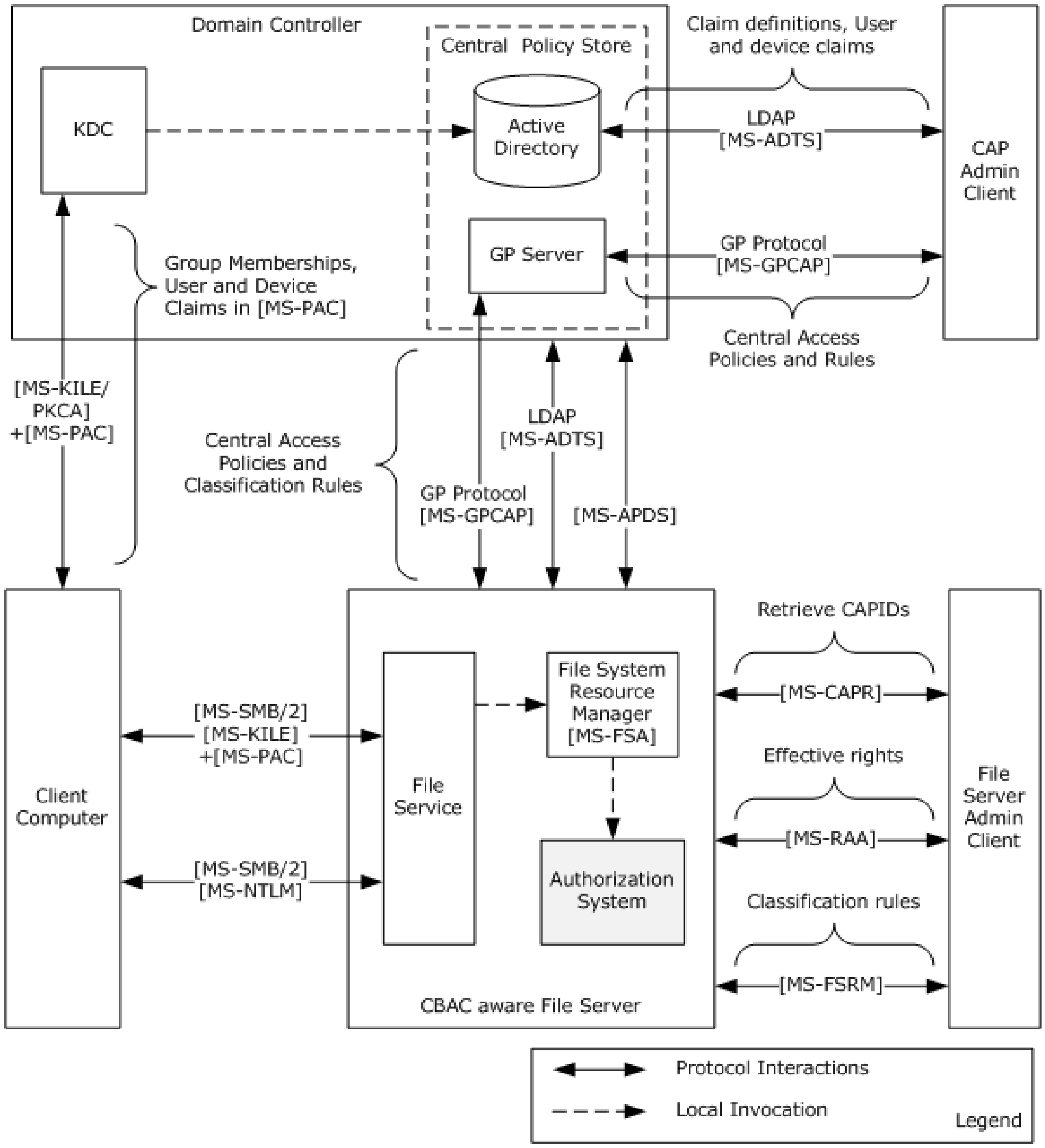

Figure 10: CBAC architecture

The CBAC architecture illustrated in the preceding figure consists of the following components:

CAP Admin Client

- Facilitates the Administrator to configure the claim definitions (indicates the claim names and types of the values) and assignment of the claims to the users and devices on Active Directory store using the LDAP protocol [MS-ADTS].
- Also facilitates the Administrator to configure the **central access rules (CAR)** and **central access policies (CAP)** on the Group Policy server using the Group Policy: Central Access Policies Protocol Extension [MS-GPCAP].

Central Policy Store

- Active Directory stores the claim definitions, user and device claims, central access rules and central access policies.
- Group Policy server pushes access rules and policies to the specified file servers via Group Policy Central Access Policies Protocol Extension (see [MS-GPCAP]).

Client Computer

- The identities of the SMB clients on the client computer can get authenticated by using either the NTLM protocol ([MS-NLMP]and [MS-APDS]) or the Kerberos Protocol Extensions ([MS-KILE] or [MS-PKCA]) as described in [MS-AUTHSOD]. The Kerberos authentication protocol results in authorization information with the claims, but whereas NTLM protocol results in authorization information without the claims.
- The SMB clients request to access a file share on a remote file server by sending authorization information which is created by successful authentication.

File Server Admin Client

- Facilitates the Administrator to configure the classification rules using the FSRM protocol interfaces (see [MS-FSRM]) and retrieval of central access policies IDs using the Central Access Policy Identifier (ID) Retrieval Protocol (see [MS-CAPR]) on the remote file server.
- File Server Administrator simulates the effective rights of the users on file shares using the Remote Authorization API Protocol (see [MS-RAA]) interfaces

File Server

- Claim definitions are pulled from Active Directory using LDAP protocol (see [MS-ADTS]) queries.
- FCI and FSRM infrastructure facilitates the transfer of the resource properties and central access policies into an object's security descriptor.
- On file access requests, the file system or object store (see [MS-FSA]) calls the authorization system to determine access to files.
- The authorization system verifies access to the files as described in [MS-DTYP] section 2.5.3.2.

2.1.4.3.1 Down-level Scenarios

The following figure depicts the protocol communications for the CBAC down-level scenario, where the user tries to access the CBAC-aware shared-file resources on the file server using a file access client (CIFS or SMB or SMB2, as described in [MS-CIFS]. [MS-SMB], and [MS-SMB2]) on the down-level client computer that is running a pre-Windows 8 operating system version. The identity of the file access client has been authenticated by the authentication services system using either KILE or PKCA and has obtained the service ticket for the remote file server with the authorization information ([MS-PAC]) of the requesting identity as described in [MS-AUTHSOD].

Because the authorization information [MS-PAC] received by the file service doesn't have the user claims in it, the file service on the server computer has to obtain the service ticket to itself on behalf of the user using the S4U2self extension described in [MS-SFU]. By obtaining the service ticket to itself on behalf of the user, the service receives the user access token from the LSA policy database by submitting authorization information (see [MS-PAC]) from the obtained service ticket, as described in section 2.1.4.2, which consists of group memberships and user claims. The access token contains the authorization information received from Kerberos S4U2Self, privileges granted to the client from the LSA policy database, and local security groups assigned to the user in the SAM account database (see section 2.1.4.2). The file service impersonates the user using this user's access token, and attempts to access the file on behalf of the user. The file system or object store [MS-FSA] invokes the access check function to verify the user access rights. The authorization system checks the desired access rights using the user's access token and the object's security descriptor as described in [MS-DTYP] section 2.5.3.2.

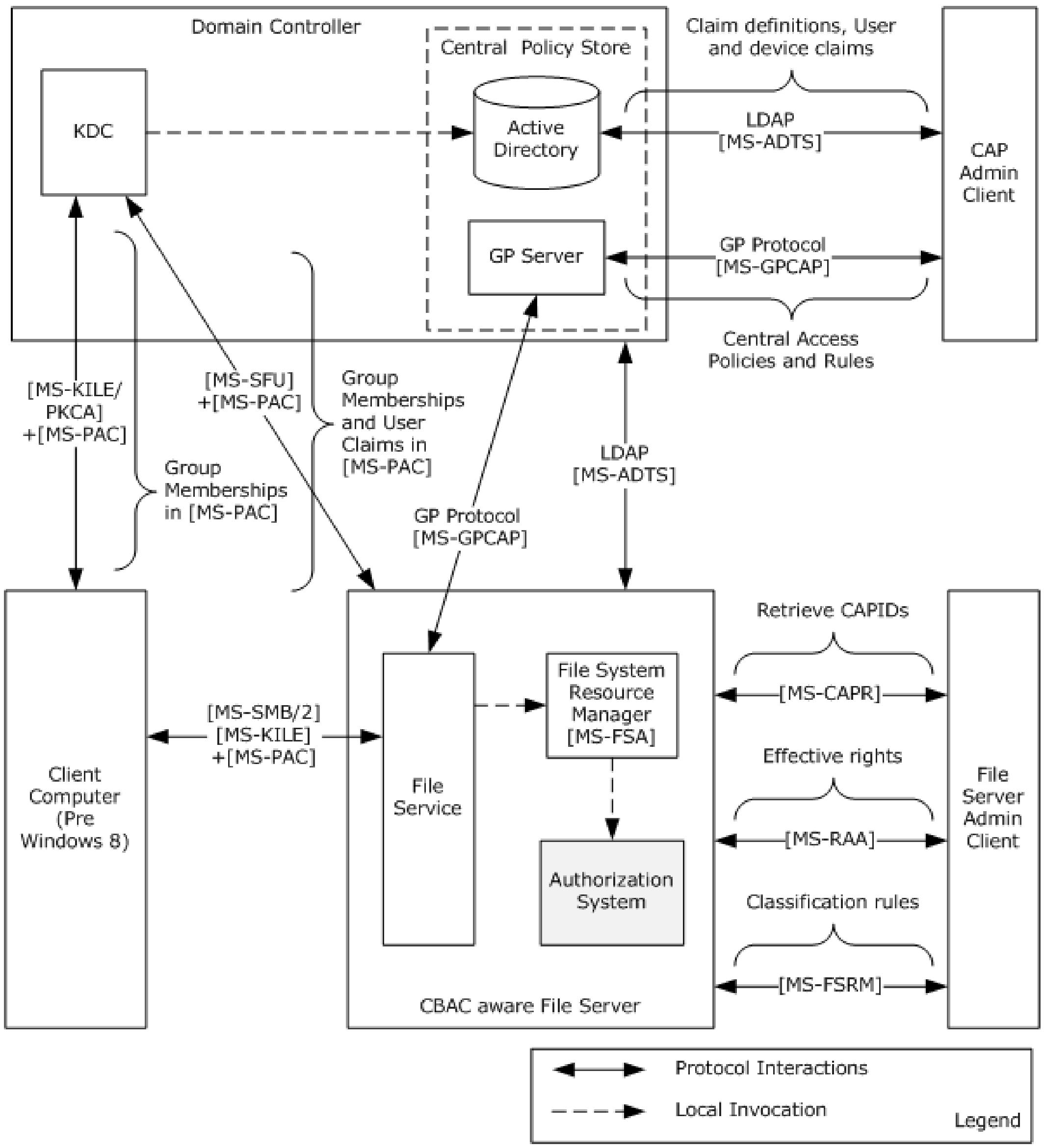

Figure 11: Protocol communications when Kerberos is the authentication protocol

The following figure depicts the protocol communications for a down level scenario, where the user tries to access the shared file resources on a Windows 8 file server computer using a file access client(SMB or SMB2) on the down level client computer that is running a pre-Windows 8 version. The identity of file access client has been authenticated by authentication services system using the NTLM protocol as described in [MS-AUTHSOD].

The process of getting the user's authorization information with the user's claims, constructing the user's access token, and verifying the access rights is the same as for Kerberos, as described earlier in this section.

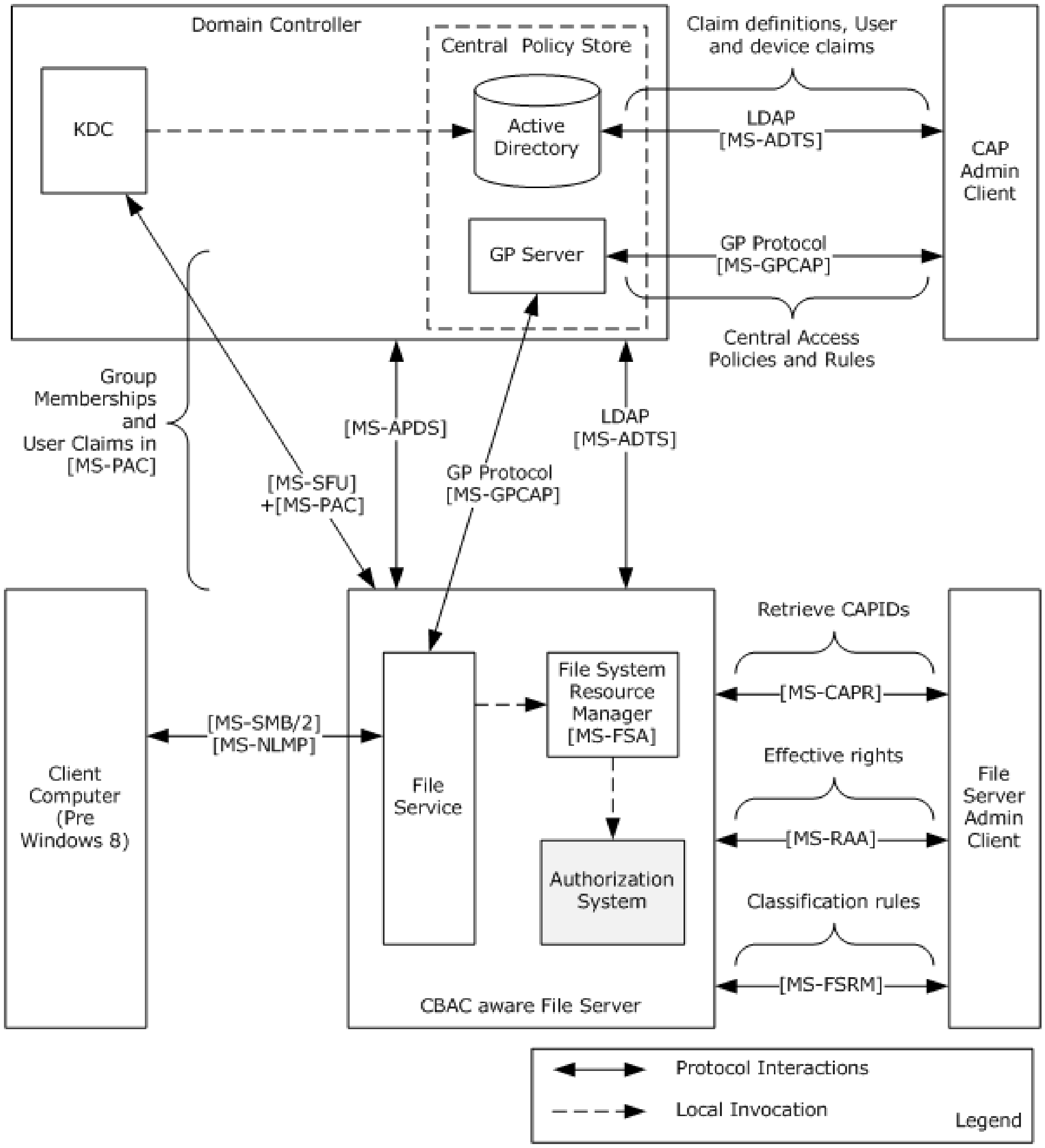

Figure 12: Protocol communications when NTLM is the authentication protocol

2.1.4.3.2 Claims Transformation

Claim type definitions are specific to a particular **forest**. In cross-forest authentication scenarios, claims need to be examined, filtered, possibly modified, and reissued when traversing from one forest to another. This process is known as claims transformation.

Claims transformation is similar in concept to SID filtering (see section 4.1.2 of MS-PAC) but more powerful. Claims transformation is based on a transformation rules grammar that administrators may use to express their intent at a fine-grained, per-claim level. The set of rules applied to incoming claims may be customized on a per-trust basis, allowing further administrator control.

The claims transformation consists of the following high-level steps:

- A PAC from a cross-realm **ticket-granting ticket (TGT)** needs to be decoded and filtered. When decoding a cross-realm TGT, the **crealm** fields inside the TGT should be compared to the expected name of the realm for the inter-realm trust. If the names do not match the TGT, they should be rejected, subject to other mitigating constraints. For more information, see MS-PAC sections 4.1.2.2 and 4.1.2.3.

- After the filtering, the next step is to obtain the claims transformation rules. This can be accomplished by using the trust name and the direction of the traversal of the trust to look up the corresponding msDS-ClaimsTransformationPolicyType object as described in [MS-ADSC] and obtain the claims transformation rules from it. For more information, see [MS-ADTS] sections 3.1.1.11.1.5 and 3.1.1.11.2.11.

- After obtaining the transformation rules, the claims to be transformed along with the transformation rules are then passed to the Claims Transformation Algorithm as described in [MS-CTA]. The output of the Claims Transformation Algorithm is further processed using the Claims Dictionary to produce claims that are relevant to the new forest in which they are used.

2.1.5 Verify Authorization

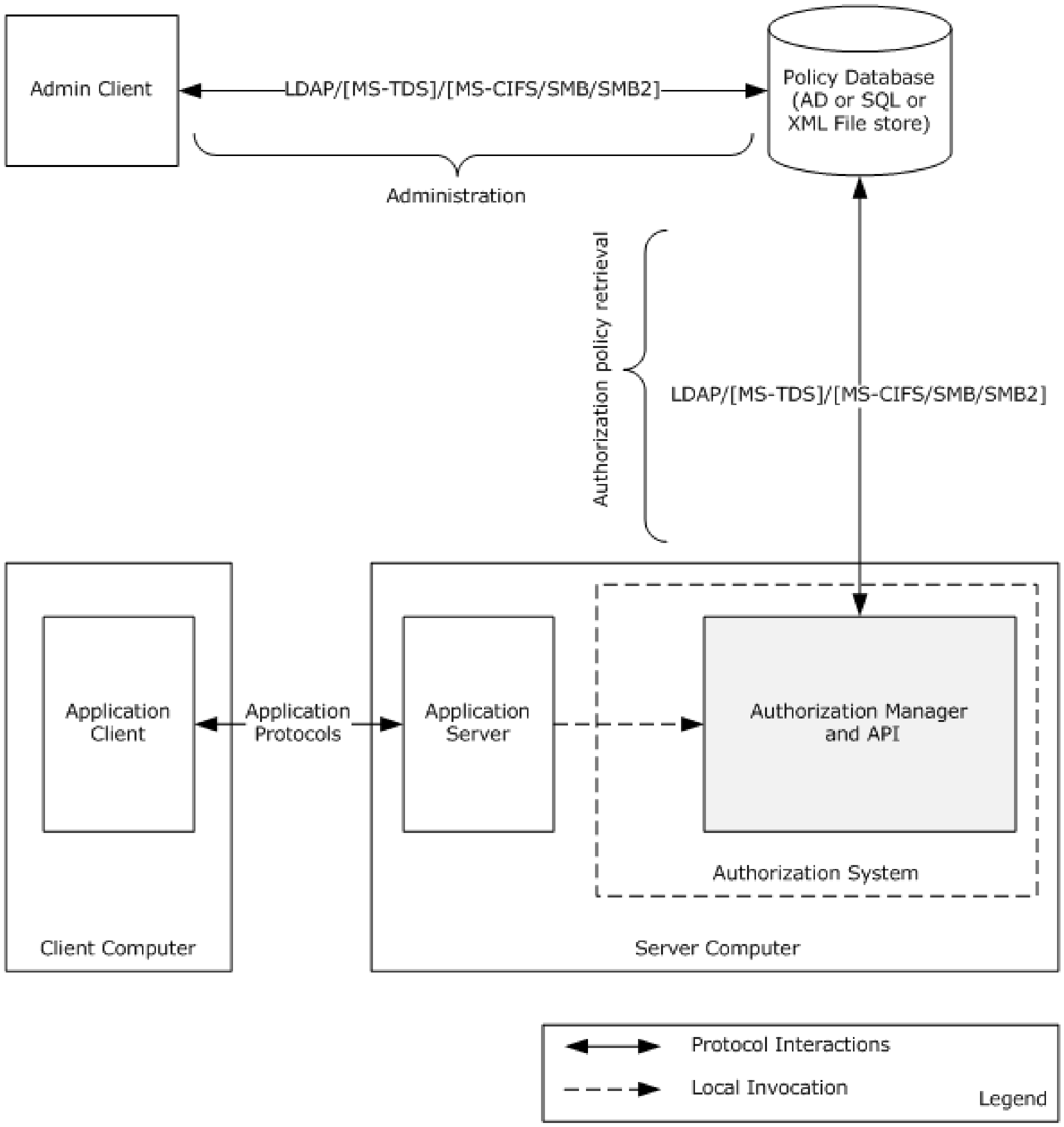

Figure 13: Authorization Manager architecture

The Authorization Manager centralized access policy database can be kept either on an **AD server**, file server, or SQL server. The Authorization Manager (AzMan) Policy File Format (see [MS-AZMP]) contains the XML schema definitions of Authorization Manager Access control policies.

The following table shows the mapping of the policy server with the corresponding protocol used.

Policy server	Protocols used
Active Directory	Lightweight Directory Access Protocol (v3) (see [MS-ADTS])
File Server	File access protocols (see [MS-CIFS], [MS-SMB], and [MS-SMB2])

Policy server	Protocols used
SQL Server	Tabular Data Stream Protocol (see [MS-TDS])

For more details on Authorization Manager, see [MSDN-AuthMgr].

2.1.6 COM+ Roles Access Control Model

The COM+ access control model implements the same set of authentication and authorization protocols that are implemented in the core DAC model.

2.1.7 Relevant Standards

None.

2.2 Protocol Summary

The following table provides a comprehensive list of the Authorization member protocols and data structures.

Protocol name	Description	Short name	Applicability
Privilege Attribute Certificate Data structure	The Privilege Attribute Certificate (PAC) is used by the authentication protocols to carry authorization information. The authorization information consists of group memberships and claims. The PAC also contains additional credential information, profile, policy information, and additional security data.	[MS-PAC]	DAC, CBAC, and COM+ roles access control
Remote Authorization API Protocol	The Remote Authorization API protocol enables applications to remotely create, query, and manipulate authorization context for a given security principal on a target server for the purpose of administrative queries. The protocol initiates creation of a security context, transfers the group and claims information, and accesses requests and result data sent between client and server.	[MS-RAA]	DAC and CBAC
Authorization Manager (AzMan) Policy File Format	The Authorization Manager (AzMan) Policy File Format contains the XML schema definitions of Authorization Manager access control policies.	[MS-AZMP]	AzMan RBAC
Group Policy Central Access Policies Protocol Extension	The Group Policy: Central Access Policies Extension is a group policy file format that communicates the Central Access Policies (CAPs) defined centrally and configured for specific computer accounts and transferred to the file servers through group policy.	[MS-GPCAP]	CBAC
Central Access Policy Identifier (ID) Retrieval Protocol Specification	This protocol enables the applications to query a remote file server for a list of Central Access Policies (CAPs) that have been configured for a remote file server. Specifically, the protocol is used to transfer the CAP IDs.	[MS-CAPR]	CBAC
Claims Transformation	This document specifies a grammar for describing transformation rule language and an algorithm for	[MS-CTA]	CBAC

Protocol name	Description	Short name	Applicability
Algorithm	transforming input claims into output claims using a defined set of rules. Transformation of a set of claims is typically used at the authentication trust traversal boundaries to transform claims from sending authority into a form acceptable to receiving authority.		
Windows Data Types	This document contains the data types and algorithms associated with authorization.	[MS-DTYP]	DAC, CBAC,and COM+ roles
Lightweight Directory Access Protocol	In CBAC: This protocol enables the applications to configure the claim definitions, and the user and devices claims on the Active Directory Server. In RBAC: This protocol enables the retrieval of authorization policies from the Active Directory Policy Server. In DAC: [MS-ADTS] section 5.1.3 specifies the authorization rules.	[MS-ADTS]	DAC, CBAC, and AzMan RBAC
Component Object Model Plus (COM+) Remote Administration Protocol	With regards to authorization, this protocol enables the administration interface for the role-based security configuration for the COM+ applications.	[MS-COMA]	COM+ roles access control
Tabular Data Stream Protocols	With regards to authorization, this protocol enables the retrieval of the authorization policies from the SQL Policy Store	[MS-TDS]	AzMan RBAC

2.3 Environment

2.3.1 Dependencies on This System

Windows components and subsystems that require making authorization decisions will depend on the authorization system. As a result, the authorization system influences a large number of systems and protocols.

The most prominent examples of protocols and systems that have a dependency on the authorization models are as follows:

DAC Model

- Active Directory (as described in [MS-ADOD])
- File System (as described in [MS-FASOD] and [MS-FSMOD])
- Registry services (as described in [MS-RRP])
- Printer Services (as described in [MS-PRSOD])

CBAC Model

- File Access Services (as described in [MS-FASOD])

AzMan RBAC Model

- Remote Desktop Services (as described in [MS-RDSOD])

COM+ Roles Access Control Model

In Windows, other than components of the COM + platform, there are no components/subsystems that depend on this model. But any enterprise application that uses the services of the COM+ platform can depend on this model.

2.3.2 Dependencies on Other Systems/Components

The authorization system depends on the following components and protocols:

- The DAC Model depends on the following components on the server computer:
  - Local Security Authority Policy Database for the user privileges and policies
  - SAM Account Database for the local user groups
- In addition to the components mentioned previously, the CBAC model depends on the following components:
  - The client implementation of the Group Policy Central Access Policies Extension Protocol [MS-GPCAP] to retrieve the central access policies and file classification rules.
  - LDAP client components to retrieve the claim definitions
  - The server implementation of Central Access Policy Identifier (ID) Retrieval Protocol [MS-CAPR] to provide the admin interface which enables the administrator to enforces the policies on file resources.
- The AzMan RBAC Model depends on LDAP, file access (CIFS), and SQL protocol components to retrieve the policies from Policy server, depending on the type of policy server.
- In addition to the dependencies mentioned under DAC model, the COM + role access control model depends on the following components:
  - Components related to the implementation of the [MS-COMA] protocol

2.4 Assumptions and Preconditions

The following assumptions and preconditions apply to this document:

- Information regarding network topology and/or addresses for the external server systems is configured or discoverable.
- One or more of the following external server systems has been set up and configured:
  - Active Directory
  - DNS Directory
  - LDAP Directory
  - Group Policy Server
- A DC has been set up and configured to support the domain infrastructure.
- The client and server machines have been joined to the domain.

- The identity of the user has been authenticated and the server application has associated the user's authorization information.
- Higher-layer protocols and service implementations are configured and running on the server systems, such as:
  - Distributed File System (DFS)
  - Group Policy
  - Network Time Protocol (NTP)
  - LDAP

2.5 Use Cases

The following table lists the use cases that span the functionality of the Authorization protocols. The grouping of use cases has been done based on authorization models.

Use case group	Use cases
DAC Model: File Server	Check Simple Access (section 2.5.1.1.2) Check ACL Inheritance Access (section 2.5.1.1.3) Check Conditional ACEs Based Access (section 2.5.1.1.4) Check Claims Based Access (section 2.5.1.1.5)
DAC Model: AD	Check Simple Access (section 2.5.1.2.2) Check Object-Specific Access (section 2.5.1.2.3) Control Access Right-Based Access (section 2.5.1.2.4) Control Validated Write-Based Access (section 2.5.1.2.5) Check Object Visibility (section 2.5.1.2.6)
AzMan RBAC Model	Verify Authorization (section 2.5.2.1)

2.5.1 DAC Model

2.5.1.1 File Server

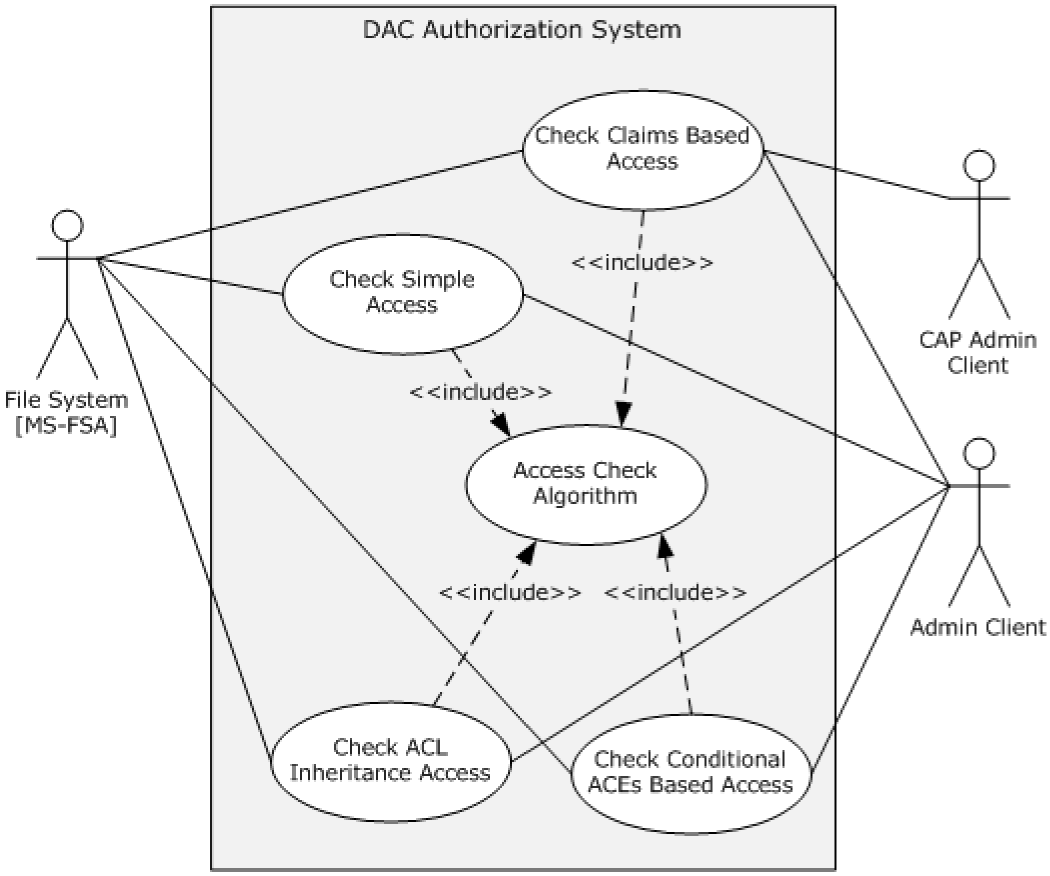

Figure 14: File server authorization use cases

2.5.1.1.1 Actors

The actors that participate in the file server DAC Model use cases are:

File system or object store: The file system implements the file system objects such as files and directories.

Admin client: The Admin client is the authorization tool that helps the Administrator to configure the access permissions on the file system objects such as files and directories

CAP Admin Client: The CAP Admin client is the administration tool that enables the administrator to configure the claim definitions, user, and device claims on Active Directory and the central access policies and classification rules on the GP server.

2.5.1.1.2 Check Simple Access

Goal

Verify the access rights of the user to access a file on a remote file share.

Context of Use

The user of the file client needs to access an existing file on a remote file share, and the file server needs to verify the access rights of the user prior to providing the access to a file. Hence, the file server interacts with the authorization system through the file system resource manager to verify the requested access rights using this use case.

Actors

Except for the CAP Admin client actor, all the actors are as described in section 2.5.1.1.1.

Stakeholders

The primary interest of a user is to access the file on the remote file server.

Preconditions

- The user of the file client has been authenticated by the Authentication Services System (see [MS-AUTHSOD]).
- The administrator using the Admin client has configured the required explicit access permissions for the requesting user to access the file on a remote file share but has not included inherited permissions from the object's parent.
- The file server obtains the access token for the requesting user as described in section 2.5.1.3, and it makes a request to the file system by passing the user's access token (which is also called security context), access rights, and other information as described in [MS-FSA] section 2.1.5.1.

Main success scenario

1. Trigger: The user tries to access an existing file on a remote file share using the file client application.
2. The file system processes the request per the processing rules described in [MS-FSA] sections 2.1.5.1 and 2.1.5.1.2.1, and these processing rules invoke the access check algorithm described in [MS-DTYP] section 2.5.3.2 to verify the access rights of the user.
3. If verification succeeds, then the access check algorithm returns success to the file system resource manager, indicating user access is allowed.

Post condition

The user of the file client is granted access to a file on remote file share.

2.5.1.1.3 Check ACL Inheritance Access

Goal

Verify the access rights of the user to access a file on a remote file share, and that the file has inheritable permissions from its parent object.

Context of Use

The user of the file client needs to access an existing file on a remote file share, and the file server needs to verify the access rights of the user prior to providing the access to a file that has both explicit access permissions and inheritable permissions from a parent object. Hence, the file server interacts with the authorization system via file system resource manager to verify the access rights of the user using this case.

Actors

Except for the CAP Admin Client actor, all the actors are as described in section 2.5.1.1.1.

Stakeholders

The primary interest of a user is to access the file on the remote file server.

Preconditions

- The user of the file client has been authenticated by the Authentication Services System (see [MS-AUTHSOD]).
- The Administrator using the Admin client has configured explicit and inherited access permissions for the requesting user to open the file on a remote file share.
- The file server obtains the access token for the requesting user as described in section 2.5.1.3, and it makes a request to the file system resource manager by passing the obtained user access token (which is also called security context), access rights, and other information as described in [MS-FSA] section 2.1.5.1.

Main success scenario

1. Trigger: The user tries to access an existing file on a remote file share using the file client application.
2. The file system processes the request per the processing rules described in [MS-FSA] sections 2.1.5.1 and 2.1.5.1.2.1, and these processing rules invoke the access check algorithm described in [MS-DTYP] section 2.5.3.2 to verify the user's access rights against the access permissions on the object's security descriptor.
3. If verification succeeds, the access check algorithm returns success to the file system resource manager, indicating user access is allowed.

Post condition

The User of the file client is granted access to a file on the remote file share.

2.5.1.1.4 Check Conditional ACEs Based Access

Goal

Verify the access rights of the user to open an existing file, on a remote file share, that has conditional ACEs configured on it.

Context of Use

The user of the file client needs to access a file on a remote file share, and the file server needs to verify the access rights of the user prior to providing the access to a file. Hence, the file server interacts with the authorization system through the file system resource manager to verify the requested access rights using this case.

Actors

Except for the CAP Admin Client actor, all the actors are as described in section 2.5.1.1.1.

Stakeholders

The primary interest of a user is to access the file on the remote file server.

Preconditions

- The user of the file client has been authenticated by the Authentication Services System (see [MS-AUTHSOD]).
- The Administrator using the Admin client has configured explicit, inherited, and conditional access permissions for the requesting user to open the file on a remote file share.
- The file server obtains the access token for the requesting user as described in section 2.5.1.3, and it makes a request to the file system resource manager by passing the obtained user access token (which is also called security context), access rights and other information as described in [MS-FSA] section 2.1.5.1.

Main success scenario

1. Trigger: The user tries to access an existing file on a remote file share using the file client application.
2. The file system processes the request as per the processing rules described in [MS-FSA] sections 2.1.5.1 and 2.1.5.1.2.1, and these processing rules invoke the access check algorithm described in [MS-DTYP] section 2.5.3.2 to verify the user's access rights against the configured access permissions on the object's security descriptor.
3. If verification succeeds, the access check algorithm returns success to the file system resource manager, indicating user access is allowed.

Post condition

The User of the file client is granted access to a file on a remote file share.

2.5.1.1.5 Check Claims Based Access

Goal

Verify the access rights of the user to access a file on a remote CBAC-aware file share.

Context of Use

The user of the file client needs to access an existing file on a remote file share, and the file server needs to verify the access rights of the user prior to providing the access to a file. Hence, the file server interacts with the authorization system through the file system resource manager to verify the requested access rights using this case.

Actors

See section 2.5.1.1.1.

Stakeholders

The primary interest of a user is to access the file on the remote file server.

Preconditions

- The identity of the user and client computer (compound identity) has been authenticated by the Authentication Services System as described in [MS-KILE] and [MS-AUTHSOD].

- The AD Administrator configured the claim definitions, user, and device claims on Active Directory using CAP Admin client tool.
- The GP Administrator configured required central access policies and classification rules for the file servers.
- The central access policies and classification rules applied to the resources of the file server.
- If the file server is in a different forest than the user, claims in PAC are transformed as described in section 2.1.4.3.2.
- Using this PAC, the file server obtains the access token (with user and device claims) for the requesting user as described in section 2.5.1.3, and it makes a request to the file's system resource manager by passing the obtained user access token (which is also called security context), access rights, and other information as described in [MS-FSA] section 2.1.5.1.

Main success scenario

1. Trigger: The user tries to open an existing file on a remote file share using the file client application.
2. The file system processes the request per the processing rules described in [MS-FSA] sections 2.1.5.1 and 2.1.5.1.2.1, and these processing rules invoke the access check algorithm described in [MS-DTYP] section 2.5.3.2 to verify the user's access rights against the configured access control permissions and central access policies in the object's security descriptor.
3. If verification succeeds, the access check algorithm returns success to the file system resource manager, indicating user access is allowed.

Post condition

The User of the file client is granted access to open a file on a remote file share.

2.5.1.2 Active Directory

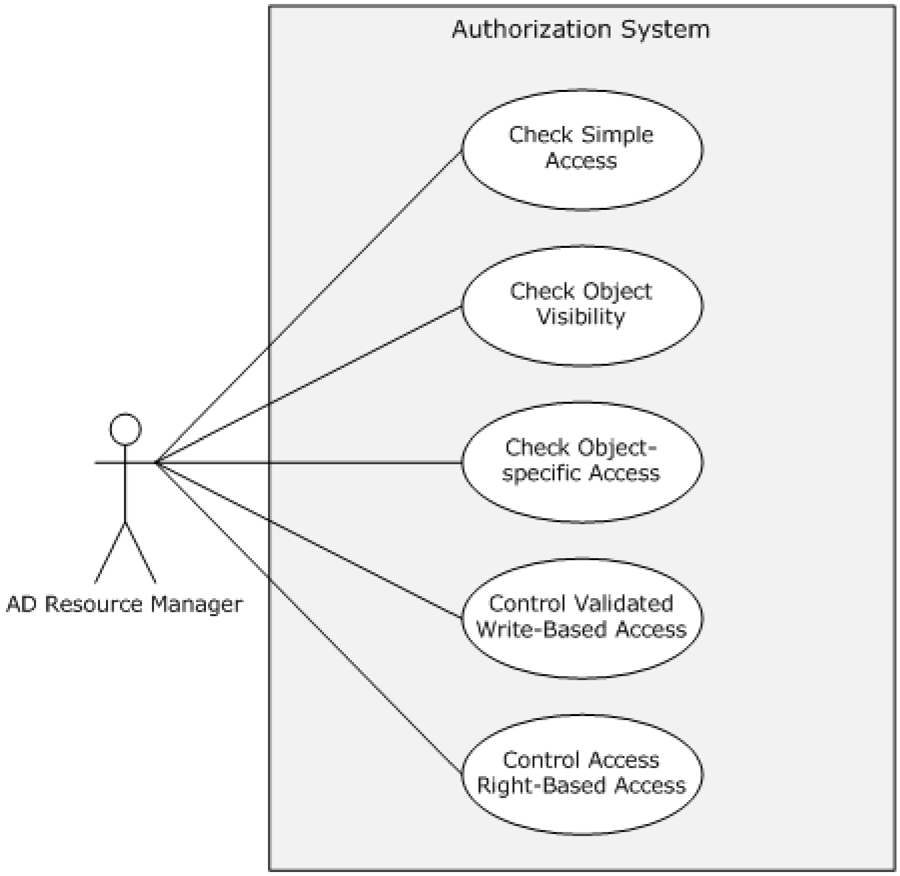

Figure 15: Active Directory authorization use cases

2.5.1.2.1 Actors

The actors that participate in the Active Directory DAC model use cases are:

Active Directory resource manager: The Active Directory resource manager ([MS-ADTS]) is code or a component that implements the active directory objects.

Admin client: The Admin client is the authorization tool that helps the Administrator to configure the access permissions for the entire Active Directory object or individual attributes of an object or the set of attributes of an object.

2.5.1.2.2 Check Simple Access

Goal

Verify the access rights of the user to access the Active Directory object on the AD server.

Context of Use

The user of the **AD client** needs to access the Active Directory object on the AD server, and the AD server needs to verify the access rights of the user prior to providing the access to the user. Hence, the AD server interacts with the authorization system through the Active Directory system resource manager to verify the requested access rights using this use case.

Actors

Same as described in section 2.5.1.2.1.

Stakeholders

The primary interest of the user of the AD client is to read all information associated with the object.

Preconditions

- The identity of the user has been authenticated by Authentication Services System [MS-AUTHSOD].
- The Administrator has configured the required access permissions for the user on the Active Directory object using the Admin tool.
- The AD server obtained the access token for the requesting user as described in section 2.5.1.3, and it already sent a request to the Active Directory resource manager by passing the user's access token (which is also called security context), access rights, and other information.
- The object's security descriptor has already undergone the SID substitution for Principal Self (see [MS-ADTS] section 5.1.3.3).

Main success scenario

1. Trigger: The user of the AD client makes request to the AD server to read all the information associated with an Active Directory object.
2. The Active Directory resource manager verifies the access rights of the user against permissions on an object's security descriptor as described in [MS-ADTS] section 5.1.3.3.2.
3. If the verification succeeds, then the Active Directory resource manager returns success to the AD server, indicating that the user is allowed to access the requested Active Directory object.

Post condition

The AD server enables access to the user to read all the information associated with the requesting Active Directory object.

2.5.1.2.3 Check Object-Specific Access

Goal

Verify the object-specific access requested by a user.

Context of Use

The user of the AD client needs to access an attribute or set of attributes on an Active Directory object, and the AD server needs to verify the user's access rights prior to allowing the access. Hence, the AD server interacts with the authorization system through the Active Directory system resource manager to verify the requested access rights using this use case.

Actors

Same as described in section 2.5.1.2.1.

Stakeholders

The primary interest of the user is to read an individual attribute of an object or a set of attributes.

Preconditions

- The identity of the user has been authenticated by the Authentication Services System [MS-AUTHSOD].
- The Administrator has configured the required attribute level access permissions for the user on the Active Directory object using the Admin tool.
- The AD server obtained the access token for the requesting user as described in section 2.5.1.3, and it already sent a request to the Active Directory resource manager by passing the user's access token (which is also called security context), access rights, and other information.
- The object's security descriptor has already undergone the SID substitution for Principal Self (see [MS-ADTS] section 5.1.3.3).

Main success scenario

1. Trigger: The user of an AD client makes a request to the AD server to read one attribute or set of attributes associated with an Active Directory object.
2. The Active Directory resource manager verifies the access rights of the user against the permissions on the object's security descriptor as described in [MS-ADTS] section 5.1.3.3.3.
3. If the verification succeeds, then the Active Directory resource manager returns success to the AD server, indicating that the user is allowed to access the requested Active Directory object.

Post condition

The AD server enables access to the user to read all the information associated with the requesting Active Directory object.

2.5.1.2.4 Control Access Right-Based Access

Goal

Verify the control access right-based access requested by the user of the AD client.

Context of Use

The user of the AD client is required to perform certain operations that have semantics that are not tied to specific properties, or where it is desirable to control access in a way that is not supported by the standard access rights; see [MS-ADTS] section 5.1.3.2.1 for more details on this. The AD server needs to verify the user's access rights prior to allowing access to perform the requested operation; hence, it interacts with the authorization system via the Active Directory resource manager to verify the requested user's access rights using this use case.

Actors

Same as described in section 2.5.1.2.1.

Stakeholders

The primary interest of a user is to perform certain operations that have semantics that are not tied to specific properties (see [MS-ADTS] section 5.1.3.2.1).

Preconditions

- The identity of the user has been authenticated by Authentication Services System [MS-AUTHSOD].
- The Administrator has configured the required attribute level access permissions for the user on the Active Directory object using the Admin tool.
- The AD server obtained the access token for the requesting user as described in section 2.5.1.3, and it already sent a request to the Active Directory resource manager by passing the user's access token (which is also called security context), the control-access-right **GUID** (see [MS-ADTS] section 5.1.3.2.1), and other information.
- The object's security descriptor has already undergone the SID substitution for Principal Self (see [MS-ADTS] section 5.1.3.3).

Main success scenario

1. Trigger: The user of an AD client makes a request to the AD server to perform the operations listed in [MS-ADTS] section 5.1.3.2.1 or extended operations provided by the application developer.
2. The Active Directory resource manager verifies the access rights of the user against permissions on the object's security descriptor as described in [MS-ADTS] section 5.1.3.3.4.
3. If the verification succeeds, the Active Directory resource manager returns success to the AD server, indicating that the user is allowed to access the requested Active Directory object.

Post condition

The AD server enables the user to perform the requested operation.

2.5.1.2.5 Control Validated Write-Based Access

Goal

Verify the write access requested by the user of the AD client to perform on attributes of an Active Directory object.

Context of Use

The user requesting attributes has configured the validated write access permissions on an Active Directory object; hence, the AD server is required to validate the values of the attributes being written. See [MS-ADTS] section 5.1.3.2.2.

Actors

Same as described in section 2.5.1.2.1.

Stakeholders

The primary interest of the user of the AD client is to write the values onto the attributes.

Preconditions

- The identity of the user has been authenticated by Authentication Services System [MS-AUTHSOD].

- The Administrator has configured the required attribute level access permissions for the user on the Active Directory object using the Admin tool.
- The AD server obtained the access token for the requesting user as described in section 2.5.1.3, and it already sent a request to the Active Directory resource manager by passing the user's access token (which is also called security context), validated rights GUID (see [MS-ADTS] section 5.1.3.2.2), and other information.
- The object's security descriptor has already undergone the SID substitution for Principal Self (see [MS-ADTS] section 5.1.3.3).

Main success scenario

1. Trigger: The user makes a request to the AD server using the AD client to get write access to an object's attributes that are controlled by validate rights.
2. The Active Directory resource manager verifies the access rights of the user against the permissions on the object's security descriptor as described in [MS-ADTS] section 5.1.3.3.5.
3. If the verification succeeds, then the Active Directory resource manager returns success to the AD server, indicating that the user is allowed to access the requested Active Directory object.

Post condition

The AD server enables the user to perform a requested write operation.

2.5.1.2.6 Check Object Visibility

Goal

Verify the access requested by the user of the AD client to enumerate the Active Directory objects and their attributes.

Context of Use

The user of the AD client needs to enumerate the Active Directory objects and their associated attributes. The AD server needs to verify the user's access rights prior to allowing the access to the client. Hence, the AD server interacts with the authorization system through the Active Directory system resource manager to verify the requested access rights using this use case.

Actors

Same as described in section 2.5.1.2.1.

Stakeholders

The primary interest of the user is to enumerate all of the Active Directory objects and their attributes.

Preconditions

- The identity of the user has been authenticated by Authentication Services System [MS-AUTHSOD].
- The Administrator has configured the required attribute level access permissions for the user on the Active Directory object using the Admin tool.

- The AD server obtained the access token for the requesting user as described in section 2.5.1.3, and it already sent a request to the Active Directory resource manager by passing the user's access token (which is also called security context), access rights, and other information.
- The object's security descriptor has already undergone the SID substitution for Principal Self (see [MS-ADTS] section 5.1.3.3).

Main success scenario

1. Trigger: The user makes a request to the AD server using the AD client to enumerate all the Active Directory objects and attributes to which the user has access.
2. The Active Directory resource manager verifies the access rights of the user against permissions on the object's security descriptor as described in [MS-ADTS] section 5.1.3.3.6.
3. If the verification succeeds, then the Active Directory resource manager returns success to the AD server, indicating that the user is allowed to access the requested Active Directory object.

Post condition

The AD server makes Active Directory objects and attributes visible to whichever user has access to them.

2.5.1.3 Auxiliary

2.5.1.3.1 Get Access Token

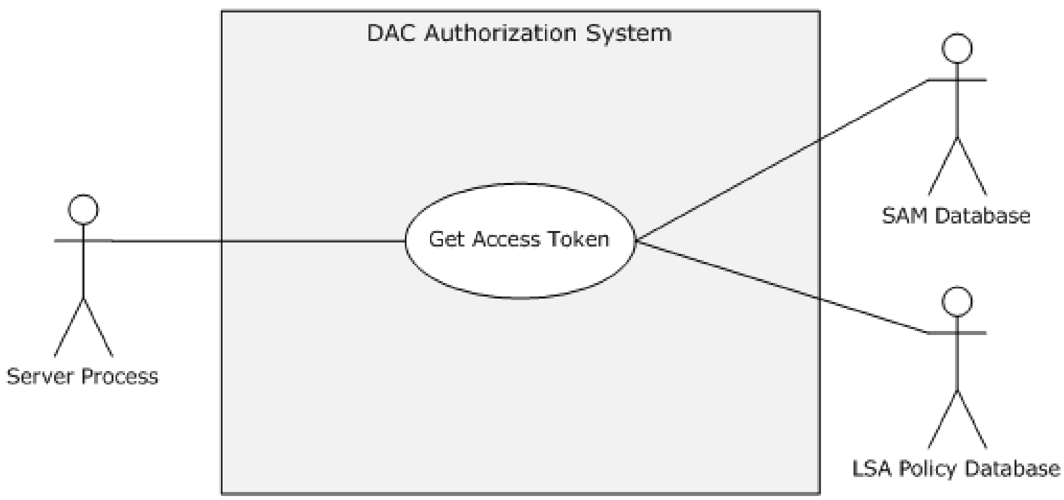

Figure 16: Get Access Token use case

Goal

Get the access token for the identity of the requestor.

Context of Use

The identity of the **application client** needs to access resources on the application server, and the application server needs to access a token to call access-check-related authorization use cases.

Actors

Application server: The application server is the service or process running on the server computer under the security context of the identity of the application server.

LSA Policy Database: A database that contains local system security policy settings such as user rights and other secrets.

SAM Database: A database that contains local users and security groups.

Stakeholders

The primary interest of the identity of the application client is to access the resources on the application server.

Preconditions

- The identity of the application client has been authenticated by the Authentication Services System (see [MS-AUTHSOD]).
- The application server has the authorization information (PAC) of the requested application client's identity.
- User rights are configured in the LSA Policy database, and local groups are configured in the SAM database.

Main success scenario

1. Trigger: The prerequisite for the application server to get the access token for authorization process.
2. The application server submits the requested identity authorization information to the authorization system.
3. The authorization system builds the access token from the User rights in the LSA Policy database and from the local security groups from the SAM database, and returns to the Application Server.

Post condition

The application server process gets the access token for the requested identity and proceeds to the next steps of the authorization process.

2.5.2 AzMan RBAC Model

2.5.2.1 AzMan RBAC Model

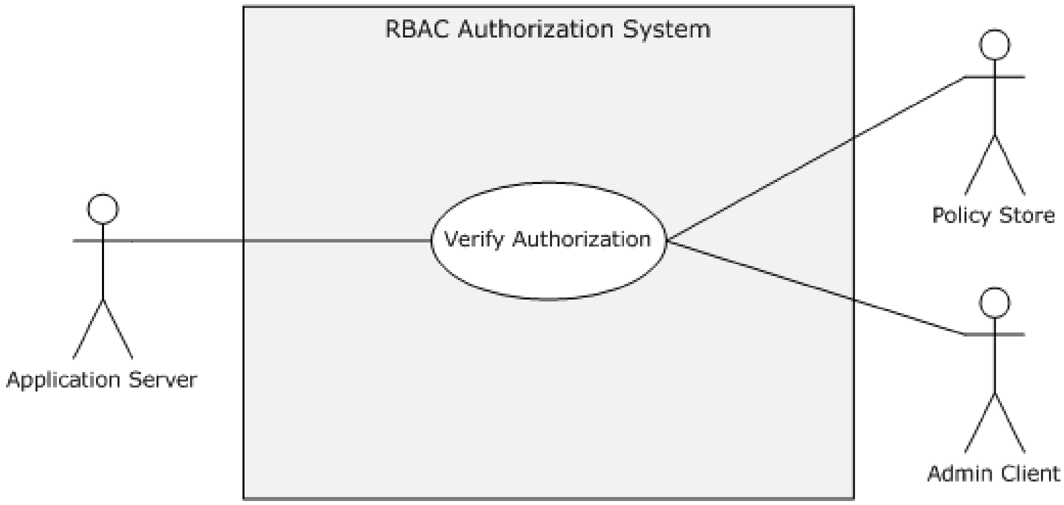

Figure 17: Verify Authorization use case

Goal

Verify the authorization rights for the user to perform the intended business operation/task.

Context of Use

The user of the application client needs to perform certain business operation/tasks using the application server, and the application server verifies the authorization of the requested user using this use case prior to allowing access to the requested business operation.

Actors

Application server: The application server is the service running on the server computer.

Admin client: The Admin client is the administrator management snap-in tool that facilitates the administrator to configure authorization policies for the applications.

Policy Store: The Policy store can be either Active Directory, SQL, or file server; it maintains the authorization policies for the applications.

Stakeholders

The primary interest of the user of the application client is to perform intended business operations/tasks with the help of the application server.

Preconditions

- The identity of the user has been authenticated, and the application server has the identity information.
- Any required authorization policies have been created on the Policy server for the application.

- The application server is configured with the required information to access the configured authorization policies.
- Required policies are configured on the Policy server for the user to perform intended business operations/tasks.

Main success scenario

1. Trigger: The user of the application client is required to perform certain protected tasks with the help of the application server.
2. The application server connects the authorization policy store with the configured details such as the connection string and gets the instance of the application policy.
3. The application server constructs the client's access token (also called security context) with the identity information of the user using Authorization Manager APIs.
4. The application server invokes the access check authorization Manager API to verify the authorization for the requested business operation/ task.

Post condition

The application server enables the user to perform requested business operation/tasks.

Extensions

None.

2.6 Versioning, Capability Negotiation, and Extensibility

There is no capability negotiation that is associated with this system. Any deviations from a specific version's implementation of these protocol specifications are documented in the respective protocol document. Capability negotiations between client and server implementations of these protocols are specified in the System Versioning and Capability Negotiation sections in their respective technical documents (TDs). For more details, see sections 1.7 of the member protocol technical documents listed in section 2.2 of this document.

2.7 Error Handling

The authorization system does not handle errors at the system level for cross-protocol error states. The individual protocol documents describe the errors that the protocols return and what they mean for the system. How to handle the errors, based on the protocol descriptions, is determined by the implementer.

2.8 Coherency Requirements

This system has no special coherency requirements.

2.9 Security

None.

2.10 Additional Considerations

None.

3 Examples

3.1 Reading from a File on Remote CBAC Aware SMB2 Share

This scenario demonstrates the use cases described in sections 2.5.1.1.5 and 2.5.1.3.1. The client and server can negotiate each other using the Simple and Protected Generic Security Service Application Program Interface Negotiation Mechanism (SPNEGO): Microsoft Extension (as described in [MS-SPNG]) to select the agreed authentication protocol as described in [MS-AUTHSOD] and [MS-SPNG].

Based on the agreed authentication protocol, this scenario has the following variants:

- Kerberos Protocol Extensions (as described in [MS-KILE] and [MS-PKCA])
- NT LAN Manager Authentication Protocol (as described in [MS-NLMP])

If the agreed authentication protocol is Kerberos, this scenario in turn has the following subvariants:

- Client has obtained a service ticket for file service from the KDC with the claims (user and device).
- Client has obtained a service ticket for file service from the KDC without the user claims.

The following are the common prerequisites of this scenario.

Common Prerequisites

- The client computer and server computer are joined to the same Active Directory domain.
- The file server and file resource manager roles have been configured on the server computer.
- The required user accounts and associated group memberships have been configured on an Account Database (see [MS-ADOD]).
- Created claim types, resource(file) properties, and central access rules (CARs) are configured on DC and then added to the central access policies using the Active Directory Administration center tool.
- The intended central access policies (CAPs) have been targeted to the file server computer using the Group Policy Management tool and the CAPs to the required file shares have been enabled.
- The required association of claims for the user and computer accounts have been set.
- Classification rules have been pushed onto the file server through the LDAP (carried File Classification Infrastructure structures ,as described in [MS-FCIADS]) Protocol.
- File share(s) have been created on the server computer and the appropriate shared permissions configured.
- The value of the ClaimsCompIdFASTSupport ADM variable on the KDC has been configured to enable claims, compound Identity, and Flexible Authentication Secure Tunneling (FAST) as described in [MS-KILE] section 3.3.1.

3.1.1 Kerberos Protocol Extensions [MS-KILE]

3.1.1.1 Service Ticket with the User and Device Claims

Prerequisites

The following are the additional prerequisites that are required for this variant, in addition to the common prerequisites mentioned in section 3.1:

- Enable Kerberos FAST on the client computer as described in [MS-KILE] section 3.2.1.
- Set the FAST-supported, Compound-identity-supported, and Claims-supported bit flags on the **msDS-SupportedEncryptionTypes** attribute of the **krbtgt** account. Refer to [MS-KILE] section 2.2.6 for details about the **msDS-SupportedEncryptionTypes** attribute.
- Set the Compound-identity-supported bit flags on the **msDS-SupportedEncryptionTypes** attribute of the File server computer account. Refer to [MS-KILE] section 2.2.6 for details about the **msDS-SupportedEncryptionTypes** attribute.

Initial System State

- The identity of the client computer account has been authenticated by the Authentication Services System as described in [MS-AUTHSOD] section 2.5.5.1, and the client computer has the TGT for the computer account.
- The identity of the user has been authenticated by KDC and the **file server**, the identity of the file server has been authenticated by the client computer as described in [MS-AUTHSOD] section 3.3.1, and the client computer has submitted the service ticket with the PAC containing group memberships, user, and device claims to access the intended file share.
- The file server has obtained the PAC with the group memberships, user, and device claims from the client, and the SMB2 client (on the client computer) has obtained the sessionId as described in the Connecting to an SMB2 Share example in [MS-AUTHSOD] section 3.3.1.
- The user running the SMB2 client application has not been authorized to the read the remote file.
- The file server has obtained the user's access token (security context) as described in section 2.5.1.3.1.

Final System State

- The user running the SMB2 client application has been authorized to read the contents of the remote file.

Sequence of Events

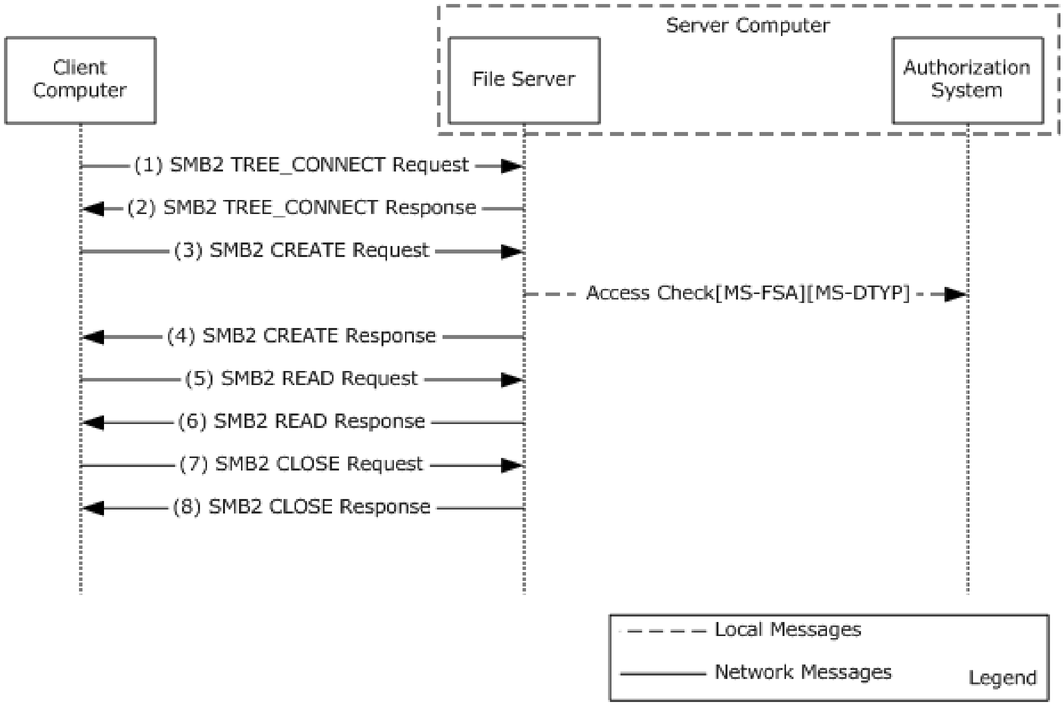

Figure 18: Reading from a file on a remote CBAC-aware SMB2 share configured with user and device claims

1. The client sends an SMB2 TREE_CONNECT Request (see [MS-SMB2] section 2.2.9) with the SessionId for the session, and a tree connect request containing the Unicode share name "\\smb2server\ShareName".

2. The server computer validates the request and verifies the access permissions on the requesting share as described in [MS-SMB2] section 3.3.5.7. If the verification succeeds, it responds with an SMB2 TREE_CONNECT response as described in [MS-SMB2]section 2.2.10.

3. The client sends and SMB2 CREATE Request (see [MS-SMB2]section 2.2.13) for the file "testfile.txt" with the appropriate access mask value (required bits for the read file operation) as described in [MS-SMB2] section 2.2.13.1.

4. The server processes the request as described [MS-SMB2] section 3.3.5.9, and makes the call to the underlying file system [MS-FSA] to verify the requesting user access rights by passing the user's access token, access rights, and other information. The file system processes the request as described in [MS-FSA] section 2.1.5.1 and invokes the access function of the authorization system to validate requesting access rights of the user. The authorization system runs the access check algorithm as described in [MS-DTYP] section 2.5.3.2 to verify the requesting access rights of the user. If the verification succeeds, the authorization system returns SUCCESS, indicating that the user is allowed to read the requesting file.

   The file server constructs an SMB2 CREATE Response (see [MS-SMB2] section 2.2.14), and responds to the client.

5. The client sends an SMB2 READ Request as described in [MS-SMB2] section 2.2.19 to read data from the file.

6. The server validates the request as described in [MS-SMB2] section 3.3.5.12. If the validation is successful, it responds with an SMB2 READ Response (see [MS-SMB2] section 2.2.20) with the data read from the file.

7. The client sends an SMB2 CLOSE Request as described in [MS-SMB2] section 2.2.15 to close the file.

8. The server sends an SMB2 CLOSE Response as described in [MS-SMB2] section 2.2.16 indicating that the close was successful.

3.1.1.2 Service Ticket Without the User Claims

This example is applicable when the client computer is pre-Windows 8 operating system and uses the Kerberos as authentication protocol.

Prerequisites

The following are the additional prerequisites that are required for this variant, in addition to the common prerequisites mentioned in section 3.1:

- The file server service has been authenticated by the KDC and has a TGT for the service account.

Initial System State

- The identity of the client computer account has been authenticated by the Authentication Services System as described in [MS-AUTHSOD] section 2.5.5.1.
- The identity of the user has been authenticated by KDC and the file server, and the identity of the file server has been authenticated by the client computer as described in [MS-AUTHSOD] section 3.3.1.
- The file server has obtained the PAC with the group memberships, but not user claims from the client, and the SMB2 client (on the client computer) has obtained the sessionId as described in the Connecting to an SMB2 Share example in [MS-AUTHSOD] section 3.3.1.
- The user running the SMB2 client application has not been authorized to the read the remote file.
- The file server has obtained the user's access token (security context) as described in section 2.5.1.3.1.

Final System State

- The user running the SMB2 client application has been authorized to read the contents of the remote file.

Sequence of Events

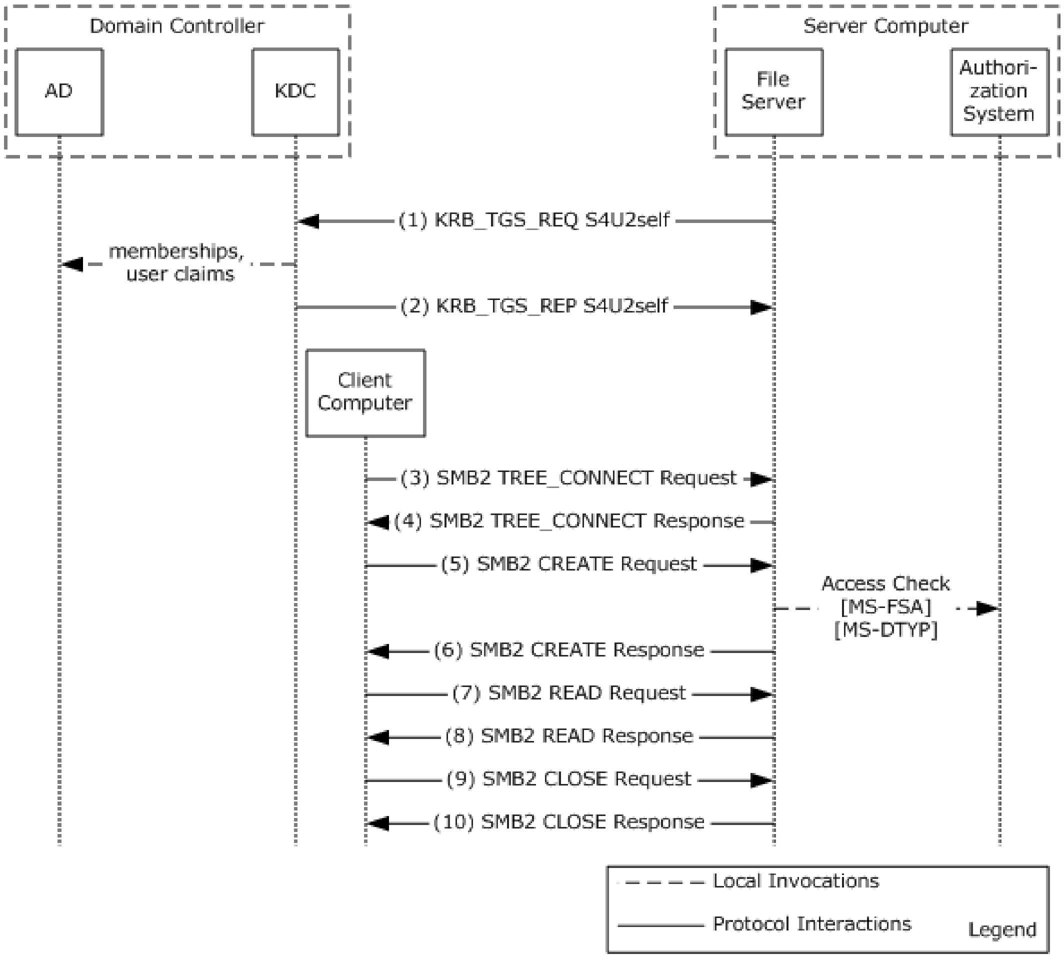

Figure 19: Reading from a file on a remote CBAC-aware SMB2 share configured with only user claims

1. The file server service uses the S4U2self extension to retrieve a user claim for itself on behalf of the user. The service fills out the **PA_FOR_USER** structure ([MS-SFU] section 2.2.1) data structure and sends the KRB_TGS_REQ message, as described in [MS-SFU] section 3.1.5.1.1, to the KDC.

2. The KDC processes the request, and retrieves the claims and group membership associated with the user from Account Database (see [MS-ADOD]) as described in [MS-SFU] section 3.2.5.1.2 and [MS-KILE] section 3.3.5.6.2.6. The KDC returns the service ticket for the user in the KRB_TGS_REP message. The privilege attribute certificate (PAC) returned in the service ticket contains the group membership information, and user claims, as specified in [MS-PAC] section 3.

3-10. Same as steps 1-8 in "Service Ticket with the User and Device Claims" variant as described in section 3.1.1.1.

3.1.2 NT LAN Manager Authentication Protocol [MS-NLMP]

Prerequisites

Same as mentioned in common prerequisites section in section 3.1.

Initial System State

- The identity of the user has been authenticated by Domain Controller as described in [MS-AUTHSOD] section 3.3.2.
- The user running the SMB2 client application has not been authorized to read the remote file.
- The file server has obtained the user's access token (security context) as described in section 2.5.1.3.1.

Final System State

- The user running the SMB2 client application has been authorized to read the contents of the remote file.

Sequence of Events

Same as described in the Service Ticket Without the User Claims example in section 3.1.1.2.

4 Microsoft Implementations

The information in this specification is applicable to the following versions of Windows:

- Windows 2000 operating system
- Windows XP operating system
- Windows Server 2003 operating system
- Windows Vista operating system
- Windows Server 2008 operating system
- Windows 7 operating system
- Windows Server 2008 R2 operating system
- Windows 8 operating system
- Windows Server 2012 operating system
- Windows 8.1 operating system
- Windows Server 2012 R2 operating system

Exceptions, if any, are noted in the following section.

4.1 Product Behavior

None.

5 Index

6 Change Tracking

This section identifies changes that were made to the [MS-AZOD] protocol document between the January 2013 and August 2013 releases. Changes are classified as New, Major, Minor, Editorial, or No change.

The revision class **New** means that a new document is being released.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements or functionality.
- An extensive rewrite, addition, or deletion of major portions of content.
- The removal of a document from the documentation set.
- Changes made for template compliance.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **Editorial** means that the language and formatting in the technical content was changed. Editorial changes apply to grammatical, formatting, and style issues.

The revision class **No change** means that no new technical or language changes were introduced. The technical content of the document is identical to the last released version, but minor editorial and formatting changes, as well as updates to the header and footer information, and to the revision summary, may have been made.

Major and minor changes can be described further using the following change types:

- New content added.
- Content updated.
- Content removed.
- New product behavior note added.
- Product behavior note updated.
- Product behavior note removed.
- New protocol syntax added.
- Protocol syntax updated.
- Protocol syntax removed.
- New content added due to protocol revision.
- Content updated due to protocol revision.
- Content removed due to protocol revision.

- New protocol syntax added due to protocol revision.
- Protocol syntax updated due to protocol revision.
- Protocol syntax removed due to protocol revision.
- New content added for template compliance.
- Content updated for template compliance.
- Content removed for template compliance.
- Obsolete document removed.

Editorial changes are always classified with the change type **Editorially updated.**

Some important terms used in the change type descriptions are defined as follows:

- **Protocol syntax** refers to data elements (such as packets, structures, enumerations, and methods) as well as interfaces.
- **Protocol revision** refers to changes made to a protocol that affect the bits that are sent over the wire.

The changes made to this document are listed in the following table. For more information, please contact protocol@microsoft.com.

Section	Tracking number (if applicable) and description	Major change (Y or N)	Change type
4 Microsoft Implementations	Modified this section to include references to Windows 8.1 operating system and Windows Server 2012 R2 operating system.	N	Content updated.