1Security

There are two distinguishable categories of security provided by COM. The first form is termed ***Activation Security***, and it dictates how new objects are started, how new and existing objects are connected to, and how certain public services, such as the *Class Table* and the *Running Object Table* are secured. The second form is ***Call Security***, which dictates how security operates at the call level between an established connection from a client to an object (server).

Aspects of the security API are necessarily platform dependent. The Windows versions are shown for reference. Complete interoperability is supported by the user of common, installable authenticators. COM on Windows will support at least Windows NT, Novell Netware, and DCE Kerberos security.

The remainder of this chapter describes these two forms of COM security in detail.

1.1Activation Security

As described in previous chapters, objects are exposed to clients either statically, by configuring a persistent registry with information about the server code the *Service Control Manager* launches to retrieve an object, or dynamically, through publishing an object, such as a class object via CoRegisterClassObject or a running object via IRunningObjectTable::Register. Accordingly, there are two aspects to activation security, one static (or automatic) form, and one dynamic form.

Activation security is automatically applied by the *Service Control Manager* of a particular machine. Upon receipt of a request to retrieve an object, the *Service Control Manager* checks the request against security information stored either within its registry or gathered dynamically from objects and stored within its internal tables.

1.1.1Registry Configuration

All *Service Control Managers* should offer a level of simple registry-driven configurability for use administering classes of a machine and for specific user accounts on that machine. The following tables contain suggested configuration variables and a description of their Win32 implementation as elaboration.

Machine Wide Settings	Use	Win32 Implementation
Allow Activation	Boolean enables and disables activation on a machine-wide basis.	HKLM[1]\Software\Network OLE\Enabled = [0 | 1]
Per-Class Security	Establishes automatic activation security for a specific class registered for use by any users on this machine.	HKCC[2]\CLSID\{…}\ActivationSecurity is secured.[3] HKCC\CLSID\{…}\FindActivationSecurityAt = {clsid} points to a class with an \ActivationSecurity secured key.
Default Class Security	Establishes automatic activation security for any classes without per-class security registered for use by any user on this machine.	HKLM\Software\Network OLE\DefaultActivationSecurity is secured.
Default ROT Security	Defines default security on objects placed in the *Running Object Table* of this machine by any user.	HKLM\Software\Network OLE\DefaultROTSecurity is secured.

[1] Shorthand for HKEY_LOCAL_MACHINE, the section of the registry containing machine-wide software configuration information. Typically holds configuration information used by server applications that are not running as a particular user but rather on behalf of the system.

[2] Shorthand for HKEY_COMMON_CLASSES, the section of the registry containing machine-wide class information (mappings between CLSID's and DLL/EXE names).

[3] Under Win32, when a key is *secured* the act of retrieving its value performs an access check against the security descriptor that guards it. Therefore, the *SCM*'s retrieval of the value of a secure key causes an implicit access check.

Per-User Settings	Use	Win32 Implementation
Allow Activation	Boolean enables and disables all activation for a particular user on this machine.	HKCU[4]\Software\Network OLE\Enabled = [0 | 1]
Per-Class Security	Establishes automatic activation security for a specific class registered for use by a particular user on this machine.	HKCR[5]\CLSID\{…}\ActivationSecurity is secured. HKCR\CLSID\{…}\FindActivationSecurityAt = {clsid} points to a class with an \ActivationSecurity secured key.
Default Class Security	Establishes automatic activation security for any classes without per-class security registered for use by a particular user on this machine.	HKCU\Software\Network OLE\DefaultActivationSecurity is secured.
Default ROT Security	Defines default security on objects placed in the *Running Object Table* of this machine by a particular user.	HKCU\Software\Network OLE\DefaultROTSecurity is secured.

1.1.2 *IActivationSecurity Interface*

The IActivationSecurity interface is exposed by objects which register themselves via CoRegisterClassObject and IRunningObjectTable::Register in order to secure access to the tables in which these objects are registered, as described above.

```
interface IActivationSecurity : IUnknown {
    HRESULT     GetSecurityDescriptor(SECURITY_DESCRIPTOR** ppSecDesc);
    };
```

IActivationSecurity::GetSecurityDescriptor

HRESULT IActivationSecurity::GetSecurityDescriptor(ppSecDesc);

Retrieves the security descriptor associated with this object. This security descriptor is used to control access to this object pointer in system-maintained tables.

Argument	Type	Description
ppSecDesc	SECURITY_DESCRIPTOR**	Location in which to return a pointer to the security descriptor for activation or binding to this object.
Returns	S_OK	Success. *ppSecDesc refers to a valid SECURITY_DESCRIPTOR.
	E_INVALIDARG	One or more arguments are invalid.

4 Shorthand for HKEY_CURRENT_USER, the section of the registry containing per-user software configuration information. Typically holds configuration information used by applications that are running on behalf of a particular user.

5 Shorthand for HKEY_CLASSES_ROOT, the section of the registry containing per-user class information (mappings between CLSID's and DLL/EXE names).

1.1.3Applying Activation Security

The following table outlines how activation security is applied to requests to the *Service Control Manager*.

Request	Action
CoGetClassObject or CoCreateInstance of a non-running class *X*	• Check “HKLM\Software\Network OLE\Enabled”. Fail the request if zero. • Check ”HKCU\Software\Network OLE\Enabled”. Fail the request if zero. • If class is registered in HKCR, follow ”HKCR\CLSID\{...}\FindActivationSecurityAt = {...}” until an ”HKCR\CLSID\{...}\ActivationSecurity” key is found. If these keys do not exist, use ”HKCU\Software\Network OLE\Default Activation Security”. Check the request against the security on this key. • Otherwise, if class is registered in HKCC, follow ”HKCC\CLSID\{...}\ FindActivationSecurityAt = {...}” until an ”HKCC\CLSID\{...}\ActivationSecurity” key is found. If these keys do not exist, use ”HKLM\Software\Network OLE\ Default Activation Security”. Check the request against the security on this key.
CoGetClassObject or CoCreateInstance of a running class *Y*	• Check “HKLM\Software\Network OLE\Enabled”. Fail the request if zero. • Check ”HKCU\Software\Network OLE\Enabled”. Fail the request if zero. • Check the request against the SECURITY_DESCRIPTOR available from CoRegisterClassObject(CLSID_Y, ...). This will be either the value returned by the class object’s IactivationSecurity::GetSecurityDescriptor at the time of CoRegisterClassObject or will have been taken from ”HKCU\Software\Network OLE\DefaultActivationSecurity” or “HKLM\Software\Network OLE\ DefaultActivationSecurity” at the time of CoRegisterClassObject if the class object did not support IActivationSecurity.
Running Object Table	• Check “HKLM\Software\Network OLE\Enabled”. Fail the request if zero. • Check ”HKCU\Software\Network OLE\Enabled”. Fail the request if zero. • Before performing any operation against a ROT entry (i.e., IRunningObjectTable::Revoke, IRunningObjectTable::IsRunning, IRunningObjectTable::GetObject, IRunningObjectTable::NoteTimeChange, IRunningObjectTable::GetTimeOfLastChange, or when including an entry in an IEnumMoniker::Next of an IEnumMoniker returned from IRunningObjectTable::EnumRunning), check the call against the SECURITY_DESCRIPTOR available from IRunningObjectTable::Register. This will be either the value returned by the object’s IActivationSecurity::GetSecurityDescriptor at the time of IRunningObjectTable::Register or will have been taken from ”HKCU\Software\Network OLE\DefaultROTSecurity” or “HKLM\Software\Network OLE\DefaultROTSecurity” at the time of IRunningObjectTable::Register if the object did not support IActivationSecurity.

1.2 Call Security

COM provides two mechanisms to secure calls. The first mechanism is similar to DCE-RPC: COM provides APIs that applications may use do their own security checking. The second mechanism is done automatically by the COM infrastructure. If the application provides some setup information, COM will make all the necessary checks to secure the application. This automatic mechanism does security checking for the process, not for individual objects or methods. If an application wants more fine grained security, it performs its own checking. However, the two mechanisms are not exclusive: an application may ask COM to perform automatic security checking and then perform its own.

COM call security services are divided into three categories: general APIs called by both clients and servers, new interfaces on client proxies, and server-side APIs and call-context interfaces. The general APIs allow the automatic security mechanism to be initialized and automatic authentication services to be registered. The proxy interfaces allow the client to control the security on calls to individual interfaces. The server APIs and interfaces allow the server to retrieve security information about a call and to impersonate the caller.

In a typical scenario, the client queries the object for IClientSecurity, which is implemented locally by the remoting layer. The client uses IClientSecurity to control the security of individual interface proxies on the object prior to making a call on one of the interfaces. When a call arrives at the server, the server may call CoGetCallContext to retrieve an IServerSecurity interface. IServerSecurity allows the server to check the client's authentication and to impersonate the client, if needed. The IServerSecurity object is valid for the duration of the call. CoInitializeSecurity allows the client to establish default call security for the process, avoiding the use of IClientSecurity on individual proxies. CoInitializeSecurity and CoRegisterAuthenticationServices allow a server to register automatic authentication services for the process.

Implementations of QueryInterace must never check ACLs. COM requires that an object which supports a particular IID always return success when queried for that IID. Aside from the requirement, checking ACLs on QueryInterface does not provide any real security. If client A legally has access to interface IFoo, A can hand it directly to B without any calls back to the server. Additionally, OLE caches interface pointers and will not call QueryInterface on the server every time a client does a query.

Each time a proxy is created, COM sets the security information to default values, which are the same values used for automatic security.

1.2.1General Call Security APIs

RPC_C_AUTHN Constants

Value	Description
RPC_C_AUTHN_LEVEL_NONE	Performs no authentication.
RPC_C_AUTHN_LEVEL_CONNECT	Authenticates only when the client establishes a relationship with the server. Datagram transports always use RPC_AUTHN_LEVEL_PKT instead.
RPC_C_AUTHN_LEVEL_CALL	Authenticates only at the beginning of each remote procedure call when the server receives the request. Datagram transports use RPC_C_AUTHN_LEVEL_PKT instead.
RPC_C_AUTHN_LEVEL_PKT	Authenticates that all data received is from the expected client.
RPC_C_AUTHN_LEVEL_PKT_INTEGRITY	Authenticates and verifies that none of the data transferred between client and server has been modified.
RPC_C_AUTHN_LEVEL_PKT_PRIVACY	Authenticates all previous levels and encrypts the argument value of each remote procedure call.

RPC_C_IMP Constants

Value	Description
RPC_C_IMP_LEVEL_ANONYMOUS	The client is anonymous to the server. The server process cannot obtain identification information about the client and it cannot impersonate the client.
RPC_C_IMP_LEVEL_IDENTIFY	The server can obtain the client’s identity. The server can impersonate the client for ACL checking but cannot access system objects as the client. This information is obtained when the connection is established, not on every call.
RPC_C_IMP_LEVEL_IMPERSONATE	The server process can impersonate the client's security context while acting on behalf of the client. This information is obtained when the connection is established, not on every call.
RPC_C_IMP_LEVEL_DELEGATE	The server process can impersonate the client's security context while acting on behalf of the client. The server process can also make outgoing calls to other servers while acting on behalf of the client. This information is obtained when the connection is established, not on every call.

CoInitializeSecurity

HRESULT CoInitializeSecurity(pSecDesc, AuthnLevel, Reserved);

Initializes the security layer.

Argument	Type	Description
pSecDesc	SECURITY_DESCRIPTOR*	This parameter contains two ACLs. The discretionary ACL indicates who is allowed to call this process and who is explicitly denied. The system ACL contains audit information. COM will write an audit entry for each account listed in the system ACL if the administrator for the machine has turned on auditing of COM calls on the machine. A NULL SACL implies no auditing. A SACL with no ACEs also implies no auditing. A NULL DACL will allow calls from anyone. A DACL with no ACEs allows no access. If the application passes a NULL security descriptor, COM will construct one that allows calls from the current user and local system. All calls will be audited. COM does not actually audit every call. It only audits new connections. COM will hold a pointer to the security descriptor until the last call to CoUninitialize completes. The descriptor and its components may be allocated any way the application desires, but it may not be freed until after the application uninitializes COM.
AuthnLevel	ULONG	This parameters defines the security level and impersonation level used by automatic security. It may contain one of the values from each of the RPC_C_AUTHN and RPC_C_IMP constants OR'd together. Additionally, the value RPC_C_AUTHN_MUTUAL may be OR'd in. This value causes the authentication service to guarantee that the client can find out the login account of the server securely. When calls arrive, they must be at least as high as the specified security level and impersonation level. If not, COM will automatically fail the call. Outgoing calls will be made at the specified security level or higher if COM has a hint from the server. The impersonation level will always be set as specified and not negotiated. Dynamic impersonation is not supported.
Reserved	void*	This parameter is reserved for future use. It must be set to NULL.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

CoQueryAuthenticationServices

HRESULT CoQueryAuthenticationServices(pcbAuthSvc, adwAuthSvc);

Returns a list of the authentication services that are installed on the machine. The list can be used as input to CoRegisterAuthenticationService. Different authentication services support different levels of security. For example, NTLMSSP does not support delegation or mutual authentication while *Kerberos* does. The application is responsible for only registering authentication services that provide the features the application needs. There is no way to query which services have been registered with CoRegisterAuthenticationService.

Argument	Type	Description
pcbAuthSvc	DWORD*	Returns a count of the authentication services supported on the machine.
adwAuthSvc	DWORD**	Returns a list of authentication services supported on the machine. The enumeration of authentication services is in rpcdce.h. Authentication services that are not currently in the enumeration may be installed on a machine without upgrading the operating system. The list is allocated by CoTaskMemAlloc. The application must free the list by calling CoTaskMemFree.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.
	E_OUTOFMEMORY	Insufficient memory to create the adwAuthSvc out-parameter.

CoRegisterAuthenticationService

HRESULT CoRegisterAuthenticationServices(cbauthSvc, asAuthSvc);

This API sets the list of authentication services COM will use to authenticate incoming calls. If a call arrives with a different authentication service, the call will fail. Registering authentication services does not prevent the arrival of unsecure calls (i.e., calls with no authentication service). This API can only be called before any interfaces are marshaled. Thus servers must call this if they want security. This call is not useful for clients (unless they are also servers).

This API can only be called once.

An application cannot call both CoInitializeSecurity and CoRegisterAuthenticationService.

Argument	Type	Description
cbAuthSvc	DWORD	Specify the number of authentication services in the list asAuthSvc.
asAuthSvc	SOLE_AUTHENTICATION_SERVICE*	An array of authentication services to register. The authentication services are enumerated in rpcdce.h. COM copies the list. If the principal name is NULL, COM will assume the current user id. A NULL principal name will work for NTLMSSP and *Kerberos*. It may or may not work for other authentication services.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

1.2.2IClientSecurity Interface

IClientSecurity gives the client control over the call-security of individual interfaces on a remote object.

All proxies generated by the COM MIDL compiler support the IClientSecurity interface. If QueryInterface for IClientSecurity fails, either the object is implemented in-process or it is remoted by a custom marshaler which does not support security (a custom marshaler may support security by offering the IClientSecurity interface to the client). The proxies passed as parameters to an IClientSecurity method must be from the same object as the IClientSecurity interface.

```
interface IClientSecurity : IUnknown {
    HRESULT      QueryBlanket(void* pProxy, DWORD* pcbAuthnSvc, SOLE_AUTHENTICATION_SERVICE* pasAuthnSvc,
        RPC_AUTH_IDENTITY_HANDLE** ppAuthInfo, DWORD* AuthnLevel);
    HRESULT      SetBlanket(void* pProxy, DWORD AuthnSvc, WCHAR* ServerPrincName, RPC_AUTH_IDENTITY_HANDLE* pAuthInfo,
        DWORD AuthnLevel, DWORD AuthzSvc);
    HRESULT      CopyProxy(void* pProxy, REFIID riid, void** ppCopy);
    };
```

IClientSecurity::QueryBlanket

HRESULT IClientSecurity::QueryBlanket(pProxy, pcbAuthnSvc, pasAuthnSvc, ppAuthInfo, AuthnLevel);

This method returns authentication information. This method is called by the client to find out what authentication information COM will use on calls made from the specified proxy.

Argument	Type	Description
pProxy	void*	This parameter indicates the proxy to query.
pcbAuthnSvc	DWORD*	This parameter indicates the number of entries in the array pasAuthSvc.
pasAuthnSvc	SOLE_AUTHENTICATION_SERVICE*	This parameter is an array of authentication service, principal name pairs. The first entry is the one that COM will use to make calls to the server. The array is allocated with CoTaskMemAlloc and the application must free it by calling CoTaskMemFree.
ppAuthInfo	RPC_AUTH_IDENTITY_HANDLE**	This parameter returns the value passed to CoSetProxyAuthenticationInfo. It may be NULL if you do not care.
AuthnLevel	DWORD*	This parameter returns the current authentication level. It may be NULL if you do not care.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.
	E_OUTOFMEMORY	Insufficient memory to create the pasAuthnSvc out-parameter.

IClientSecurity::SetBlanket

HRESULT IClientSecurity::SetBlanket(pProxy, AuthnSvc, ServerPrincName, pAuthInfo, AuthnLevel, AuthzSvc);

This method sets the authentication information that will be used to make calls on the specified proxy. The values specified here override the values chosen by automatic security. Calling this method changes the security values for all other users of the specified proxy. Use IClientSecurity::CopyProxy to make a private copy.

By default the authentication service and principal name is set to a list of authentication service and principal name pairs that were registered on the server. When this method is called COM will forget the default list. By default COM will try one principal name from the list of authentication services available on both machines. It will not retry if that principal name fails.

If pAuthInfo is not set, it defaults to the logged in id. AuthnLevel and AuthzSvc default to the values specified to CoInitializeSecurity. If CoInitializeSecurity is not called, they default to RPC_C_AUTHN_LEVEL_NONE and RPC_C_AUTHZ_NONE.

Security information will often be ignored if set on local interfaces. For example, it is legal to set security on the IClientSecurity interface. However, since that interface is supported locally, there is no need for security. IUnknown and IMultiQuery are special cases. The local implementation makes remote calls to support these interfaces. The local implementation will use the security settings for those interfaces.

Argument	Type	Description
pProxy	void*	This parameter indicates the proxy to set.
AuthnSvc	DWORD	This parameter indicates the authentication service. It may be RPC_C_AUTHN_NONE if no authentication is required. It may be RPC_C_AUTHN_DONT_CHANGE if you do not want to change the current value.
ServerPrincName	WCHAR*	This parameter indicates the server principal name. It may be NULL if you don't want to change the current value.
pAuthInfo	RPC_AUTH_IDENTITY_HANDLE*	This parameter sets the identity of the client. It is authentication service specific. Some authentication services allow the application to pass in a different user name and password. COM keeps a pointer to the memory passed in until COM is uninitialized or a new value is set. If NULL is specified COM uses the current identity (whether the logged in or impersonated id).
AuthnLevel	DWORD	This parameter specifies the authentication level. It may be RPC_C_AUTHN_LEVEL_DONT_CHANGE if you do not want to change the current value.
AuthzSvc	DWORD	This parameter specifies the authorization level. It may be RPC_C_AUTHZ_DONT_CHANGE is you do not want to change the current value.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments is invalid.

IClientSecurity::CopyProxy

HRESULT IClientSecurity::CopyProxy(pProxy, riid, ppCopy)

This method makes a copy of the specified proxy. Its authentication information may be changed without affecting any users of the original proxy. The copy has the default values for the authentication information. The copy has one reference and must be released.

Local interfaces may not be copied. IUnknown, IMultiQuery, and IClientSecurity are examples of existing local interfaces.

Argument	Type	Description
pProxy	void*	This parameter indicates the proxy to copy.
riid	REFIIID	Identifies the proxy to return.
ppCopy	void**	The copy is returned to this parameter.
Returns	S_OK	Success.
	E_NOINTERFACE	The interface riid is not supported by this object.

1.2.3Client APIs for Call Security

CoQueryProxyAuthenticationInfo

HRESULT CoQueryProxyAuthenticationInfo(pProxy, pcbAuthnSvc, pasAuthnSvc, ppAuthInfo, pAuthnLevel);

Returns the authentication information used to make calls on the specified proxy. This function encapsulates the following sequence of common calls:

```
pProxy->QueryInterface(IID_IClientSecurity, (void**)&pcs);
pcs->QueryBlanket(pProxy, AuthnSvc, ServerPrincName, pAuthInfo, AuthnLevel);
pcs->Release();
```

Argument	Type	Description
see IClientSecurity::QueryBlanket		

CoSetProxyAuthenticationInfo

HRESULT CoSetProxyAuthenticationInfo(pProxy, AuthnSvc, ServerPrincName, pAuthInfo,
AuthnLevel, AuthzSvc);

Sets the authentication information that will be used to make calls on the specified proxy. This function encapsulates the following sequence of common calls:

```
pProxy->QueryInterface(IID_IClientSecurity, (void**)&pcs);
pcs->SetBlanket(pProxy, AuthnSvc, ServerPrincName, pAuthInfo, AuthnLevel);
pcs->Release();
```

Argument	Type	Description
see IClientSecurity::SetBlanket		

CoCopyProxy

HRESULT CoCopyProxy(pProxy, riid, ppCopy);

Makes a copy of the specified proxy. This function encapsulates the following sequence of common calls:

```
pProxy->QueryInterface(IID_IClientSecurity, (void**)&pcs);
pcs->CopyProxy(pProxy, riid, ppCopy);
pcs->Release();
```

Argument	Type	Description
see IClientSecurity::CopyProxy		

1.2.4IServerSecurity Interface

IServerSecurity may be used to impersonate the client during a call, even on other threads within the server. IServerSeciruty:QueryBlanket and IServerSecurity::ImpersonateClient may only be called before the call completes. IServerSecurity::RevertToSelf may be called at any time. The interface pointer must be released when it is no longer needed. Unless the server wishes to impersonate the client on another thread, there is not reason to keep an IServerSecurity past the end of the call, since it will at that point no longer support IServerSecurity::QueryBlanket.

```
interface IServerSecurity : IUnknown {
    HRESULT     QueryBlanket(RPC_AUTHZ_HANDLE* Privs, WCHAR** ServerPrincName, DWORD* AuthnLevel, DWORD* AuthnSvc,
        DWORD* AuthzSvc );
    HRESULT     ImpersonateClient(void);
    HRESULT     RevertToSelf(void);
    };
```

IServerSecurity::QueryBlanket

HRESULT IServerSecurity::QueryBlanket(Privs, ServerPrincName, AuthnLevel, AuthnSvc, AuthzSvc);

This method is used by the server to find out about the client that invoked one of its methods. CoGetCallContext with IID_ISeverSecurity returns an IServerSecurity interface for the current call on the current thread. This interface pointer may be used on any thread and calls to it may succeed until the call completes.

Argument	Type	Description
Privs	RPC_AUTHZ_HANDLE*	Returns a pointer to a handle to the privilege information for the client application. The format of the structure is authentication service specific. The application should not write or free the memory. The information is only valid for the duration of the current call. NULL may be passed if the application is not interested in this parameter.
ServerPrincName	WCHAR*	This parameter indicates the principal name the client specified. It is a copy allocated with CoTaskMemAlloc. The application must call CoTaskMemFree to release it. NULL may be passed if the application is not interested in this parameter.
AuthnLevel	DWORD*	This parameter indicates the authentication level. NULL may be passed if the application is not interested in this parameter.
AuthnSvc	DWORD*	This parameter indicate the authentication service the client specified. NULL may be passed if the application is not interested in this parameter.
AuthzSvc	DWORD*	This parameter indicates the authorization service. NULL may be passed if the application is not interested in this parameter.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.
	E_OUTOFMEMORY	Insufficient memory to create one or more out-parameters.

IServerSecurity::ImpersonateClient

HRESULT IServerSecurity::ImpersonateClient();

This method allows a server to impersonate a client for the duration of a call. The server may impersonate the client on any secure call at identify, impersonate, or delegate level. At identify level, the server may only find out the clients name and perform ACL checks; it may not access system objects as the client. At delegate level the server may make off machine calls while impersonating the client. The impersonation information only lasts till the end of the current method call. At that time IServerSecurity::RevertToSelf will automatically be called if necessary.

Impersonation information is not normally nested. The last call to any Win32 impersonation mechanism overrides any previous impersonation. However, in the apartment model, impersonation is maintained during nested calls. Thus if the server ***A*** receives a call from ***B***, impersonates, calls ***C***, receives a call from ***D***, impersonates, reverts, and receives the reply from ***C***, the impersonation will be set back to ***B***, not ***A***.

If IServerSecurity::ImpersonateClient is called on a thread other then the one that received the call, the impersonation will not automatically be revoked. It will be valid past the end of the call. However, IServerSecurity::ImpersonateClient must be called before the original call completes.

Argument	Type	Description
Returns	S_OK	Success.
	E_FAIL	The caller can not impersonate the client identified by this ISeverSecurity object.

IServerSecurity::RevertToSelf

HRESULT IServerSecurity::RevertToSelf();

This method restores the authentication information on a thread to the process's identity.

In the apartment model, IServerSecurity::RevertToSelf only affects the current method invocation. If there are nested method invocations, they each may have there own impersonation and COM will correctly restore the impersonation before returning to them (regardless of whether or not IServerSecurity::RevertToSelf was called).

IServerSecurity::RevertToSelf may be called on threads other then the one that received the call. IServerSecurity::RevertToSelf may be called after the call completes. Calls to IServerSecurity::RevertToSelf that are not matched with an IServerSecurity::ImpersonateClient call will fail.

Argument	Type	Description
Returns	S_OK	Success.
	E_FAIL	This call was not preceded by a call to IServerSecurity::ImpersonateClient on this thread of execution.

1.2.5Sever APIs for Call Security

The following APIs are provided to give the server access to any contextual information of the caller and to encapsulate common sequences of security checking and caller impersonation.

CoGetCallContext

HRESULT CoGetCallContext(riid, ppv);

Retrieves the context of the current call on the current thread. riid specifies the interface on the context to retrieve. Currently only IServerSecurity is available from the default call-context (see ISeverSecurity for details).

Argument	Type	Description
riid	REFIID	Identifies the interface to return.
ppv	void**	Returns an interface for the current call.
Returns	S_OK	Success.
	E_NOINTERFACE	The call context does not support the interface identified by riid.

CoSetCallContext

HRESULT CoSetCallContext(punk);

Establishes the call context for the current call, overriding the default call context object normally available via CoGetCallContext.

This function is provided primarily for objects performing custom marshaling. Before transferring control from their stub or IPC mechanism to the server-side code, a custom marshaler may establish the call context via CoSetCallContext so that subsequent objects can be written to take advantage of call-level security or other caller-specific contextual information in a transport neutral fashion, e.g. without regard to whether an object between them and the client was remoted via custom marshaling.

The call context reverts automatically at the end of each call. Furthermore, a custom marshaling layer which calls CoSetCallContext prior to entering the server need not call CoSetCallContext(NULL) after each returning call.

A second call to CoSetCallContext with a non-NULL punk will Release the first punk and AddRef the second.

Argument	Type	Description
punk	IUnknown*	When non-NULL, the IUnknown which is to be QueryInterface'd for the requested call context interface by subsequent calls to CoGetCallContext during the span of the current call. This interface is AddRef'd prior to returning. When NULL, resets the call context to the COM-provided default for the current call.
Returns	S_OK	Success.
	E_INVALIDARG	One or more arguments are invalid.

CoQueryClientAuthenticationInfo

HRESULT CoQueryClientAuthenticationInfo(Privs, ServerPrincName, AuthnLevel, AuthnSvc, AuthzSvc);

Used by the server to find out about the client that invoked the method executing on the current thread. This function encapsulates the following sequence of common calls:

```
CoGetCallContext(IID_IServerSecurity, (void**)&pss);
pss->QueryBlanket(Privs, ServerPrincName, AuthnLevel, AuthnSvc, AuthzSvc);
pss->Release();
```

Argument	Type	Description

see IServerSecurity::QueryBlanket

CoImpersonateClient

HRESULT CoImpersonateClient();

Allows the server to impersonate the client of the current call for the duration of the call. This function encapsulates the following sequence of common calls:

```
CoGetCallContext(IID_IServerSecurity, (void**)&pss);
pss->ImpersonateClient();
pss->Release();
```

Argument	Type	Description

see IServerSecurity::ImpersonateClient

CoRevertToSelf

HRESULT CoRevertToSelf();

Restores the authentication information on a thread of execution to its previous identity. This function encapsulates the following sequence of common calls:

```
CoGetCallContext(IID_IServerSecurity, (void**)&pss);
pss->RevertToSelf();
pss->Release();
```

Argument	Type	Description

see IServerSecurity::RevertToSelf