A Process for Establishing Due Diligence in the Protection of Wintel Based Information Systems

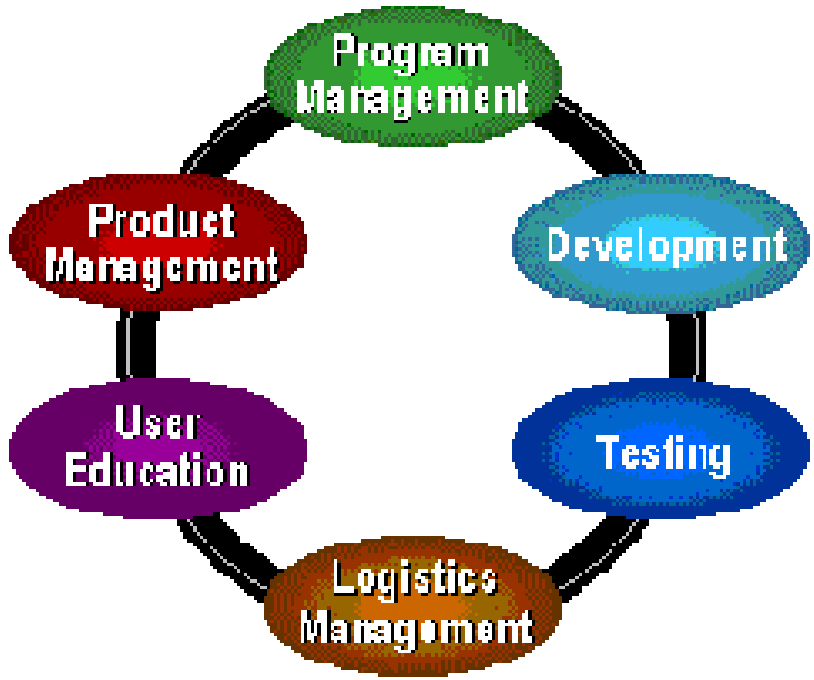
-- Bill Martin

MSF Process and Security

- Leverage opportunities to improve security using the new Windows2000 operating system.
- follow the Microsoft Solutions Framework (MSF) as a means of change control.
- include a security plan, a security risk tracking tool, and formal written documentation of the cost/benefit tradeoff decisions pertaining to each security vulnerability.
- compare the final implementation with written company security policy and standards.
- Gaps (or variances) between policy and standards should be expressly identified.
- plan to reassess existing vulnerabilities within a definite period of time.
- plan for partial mitigation of the vulnerability, even if that plan is only increased managerial oversight or regular review of data or audit logs.

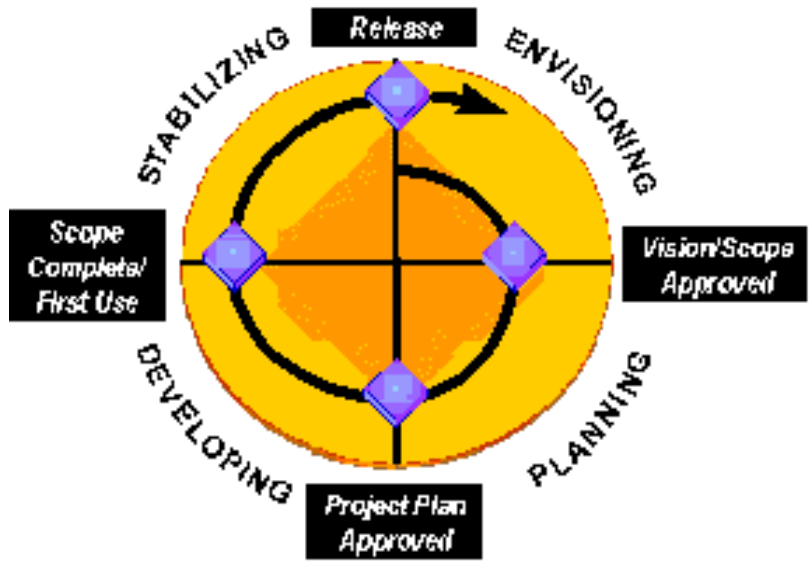
Microsoft Solutions Framework

- The MSF Team Model is comprised of six clearly defined roles:
- Microsoft® Solutions Framework (MSF) is a suite of models, principles, and guides.



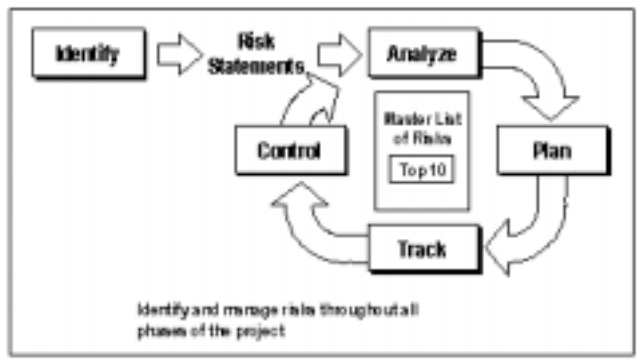
The MSF Process Model

- The MSF Process Model consists of four major milestones.
- Deliverables at each of these milestones are placed under change control when the milestone is achieved.
- Change control is a formal procedure for gaining consensus on changes to stable documents or code.



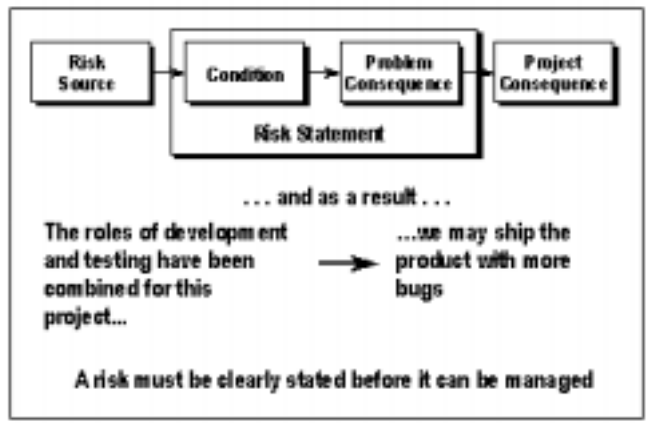
MSF Risk Management Process

- A discipline and environment of proactive decisions and actions to assess continuously what can go wrong, determine what risks are important to deal with, and implement strategies to deal with those risks.
- carry the risks forward and deal with them until they are resolved or until they turn into problems and are handled as such.



The statement of risk

- Before a risk can be managed, it must be expressed clearly.
- the team must consider not only a symptom, but also a result.
- statement of risk should include condition causing the situation to arise and the consequence).



The Risk Statement Form

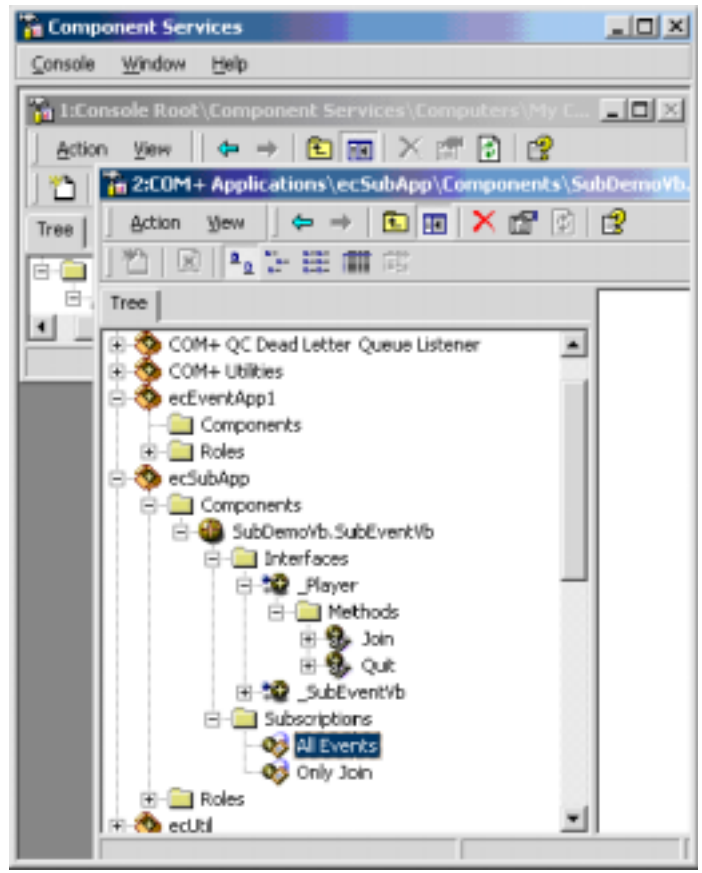
- Risk identifier.
- Risk source.
- Risk condition.
- Risk consequence.
- Risk probability.
- Risk impact classification.
- Risk impact.
- Risk exposure.
- Risk context.
- Related risks.

The Risk Action Form

- Risk identifier.
- Risk statement.
- Risk management strategy.
- Risk management strategy metrics.
- Action items.
- Due dates.
- Personnel assignments.
- Risk contingency strategy.
- Risk contingency strategy metrics and trigger values.

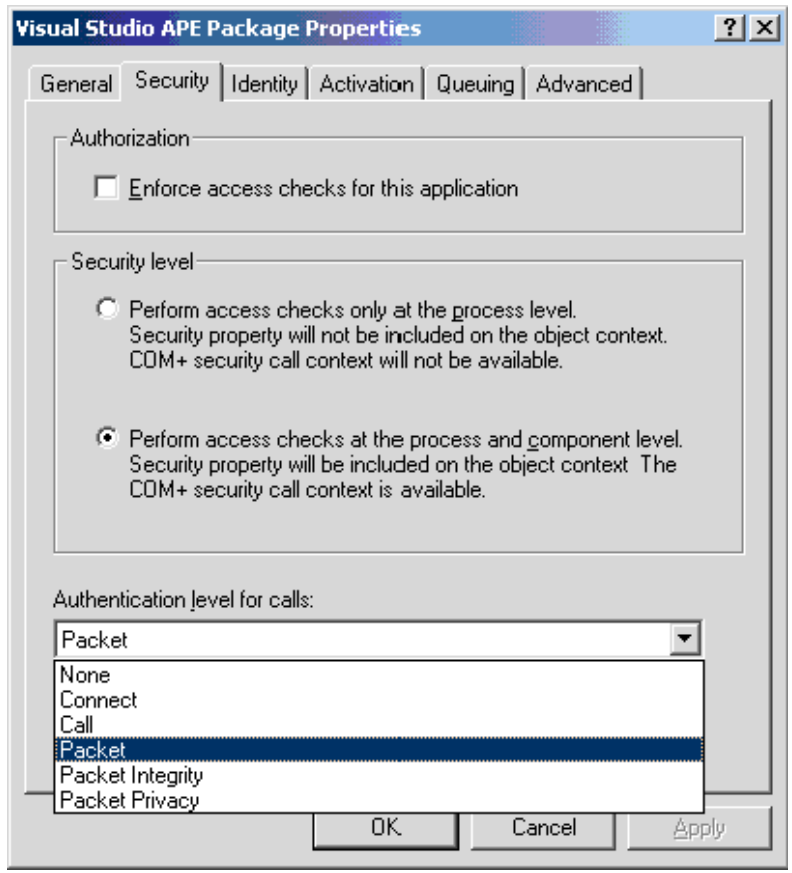
COM+ Infrastructure

- Basic security tasks done by configuration
- COM+ interfaces allow elaborate security coding.



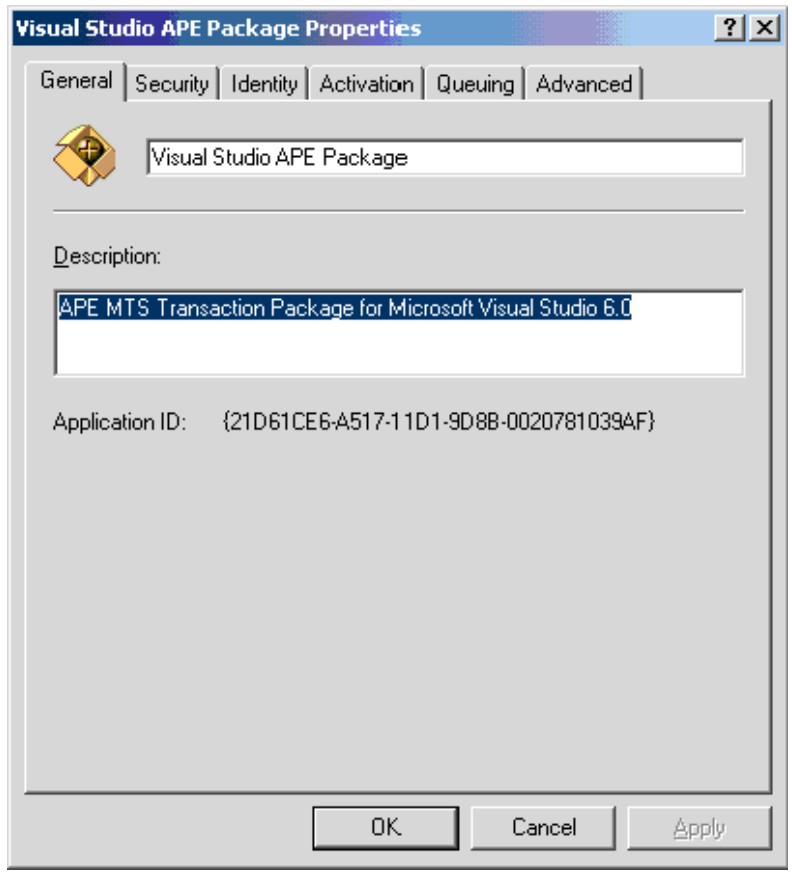
Call Authentication

- Can authenticate each and every packet.
- Prevent packet manipulation enroute
- packet privacy causes contents to be encrypted



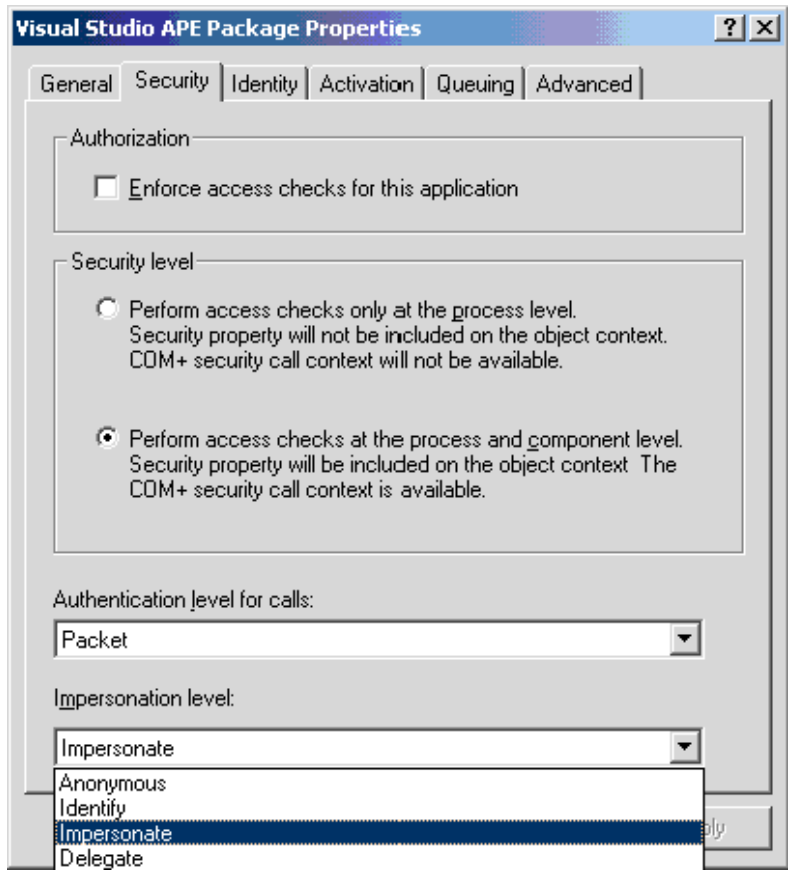
GUIDs used liberally

- Application packages are identified by GUID.
- COM+ objects each identified by GUID
- GUID based security.



Levels of impersonation

- Application can run on behalf of server anonymously
- App can impersonate the client
- Delegate option: impersonation carries forward across multiple applications

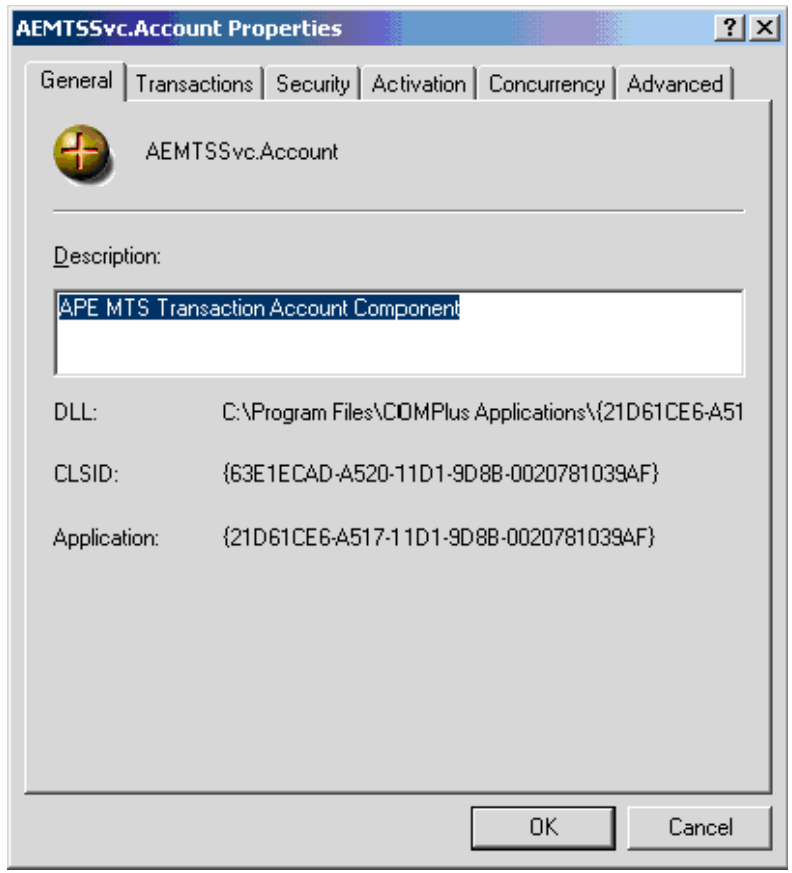


Run Identity

- App identity should be a specific user
- For debugging can use interactive user
- Facilitates logging to a log file.

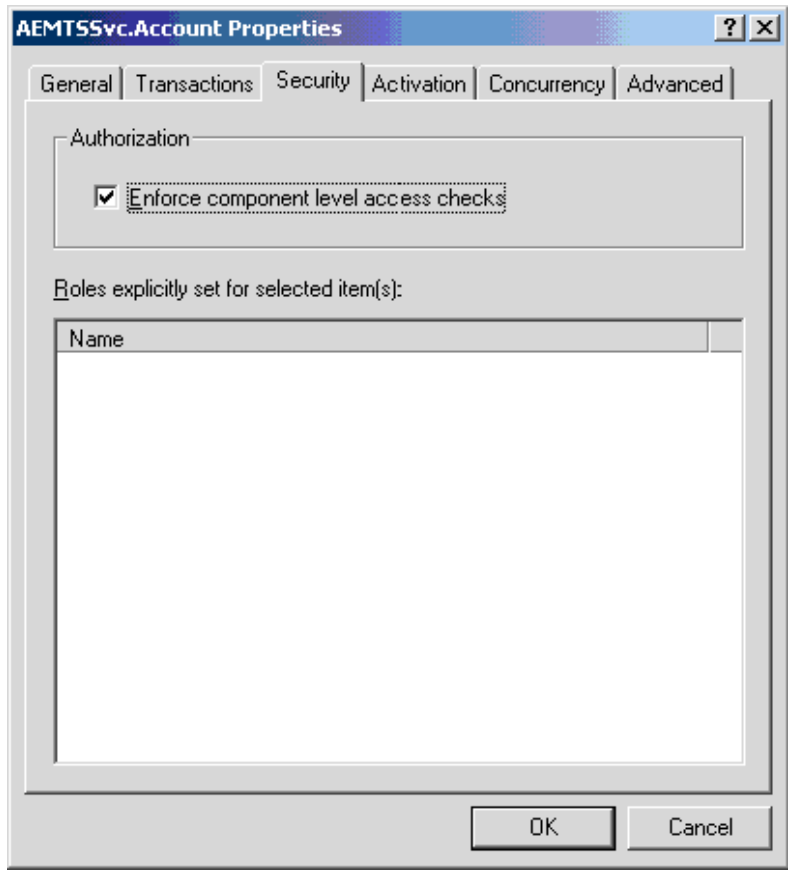
Component Properties

- DLL GUID
- Class ID by GUID
- Application ID by GUID



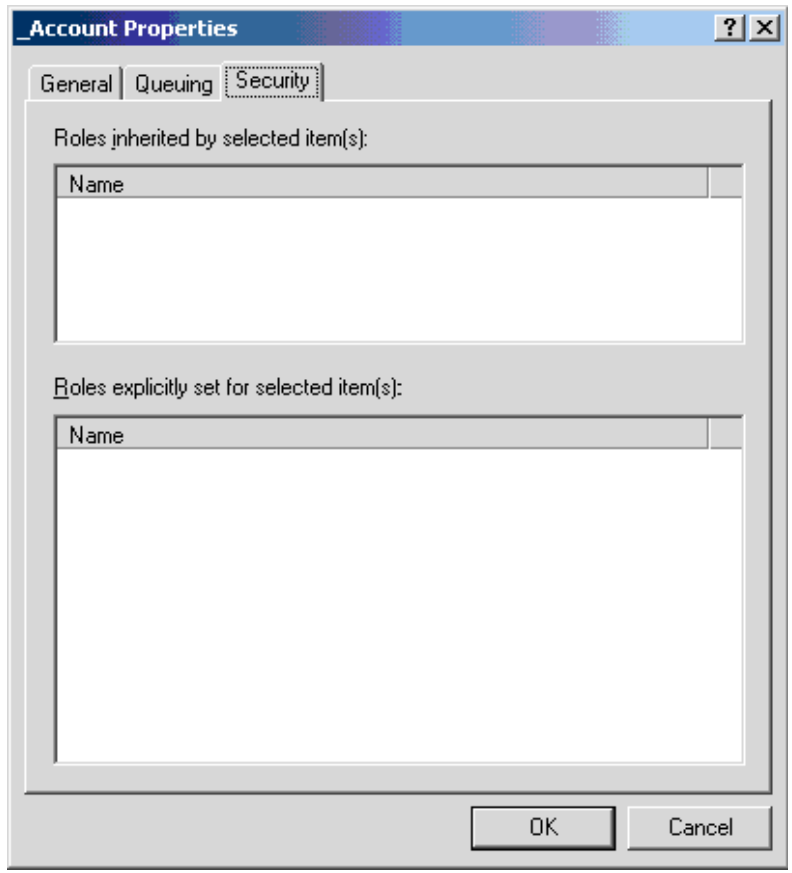
Component Access Checks

- Component level access checks can be enforced.
- Roles can be set for component use authorization.



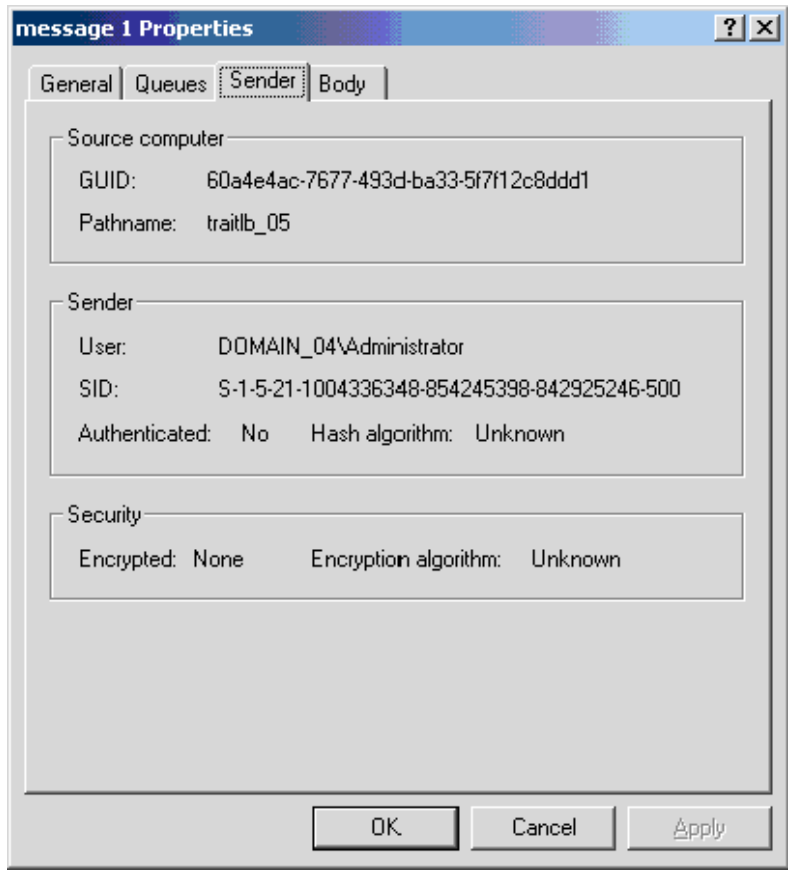
Interface Method Configuration

- Roles can be inherited by a component
- Roles can be set for use of the interface method.



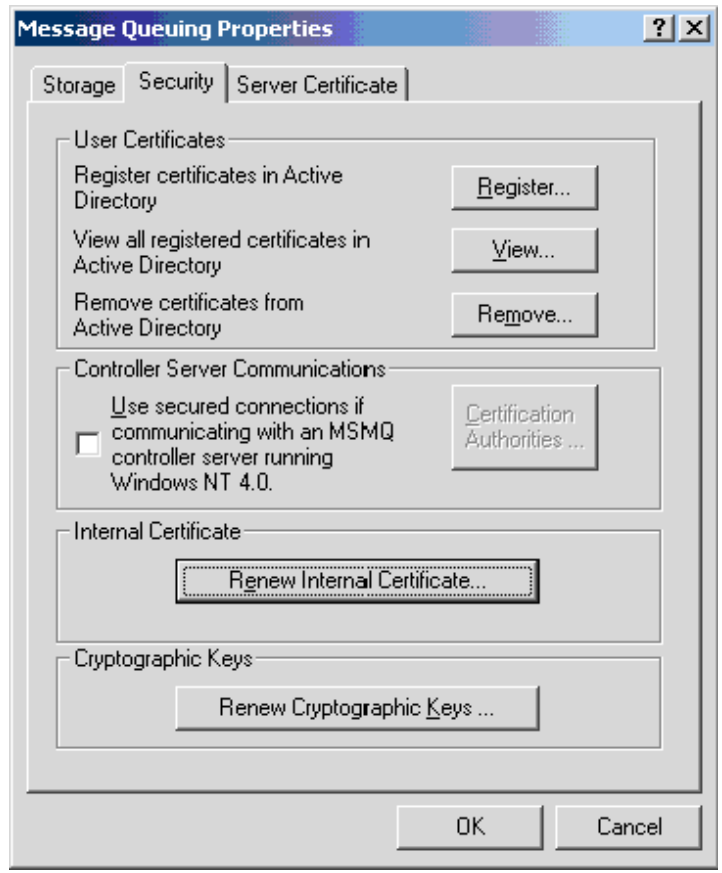
MSMQ Messages

- Middleware messages can also be secured in a Microsoft COM+ infrastructure.
- Each user can have an internal MSMQ certificate to be able to use queued components.



Message queueing security

- MSMQ internal security certificate for each user account expected to use queued components.



SECURITY ATTRIBUTES

- The **SECURITY_ATTRIBUTES** structure contains the security descriptor for an object and specifies whether the handle retrieved by specifying this structure is inheritable.

```
typedef struct _SECURITY_ATTRIBUTES {
  DWORD  nLength <>;
  LPVOID lpSecurityDescriptor <>;
  BOOL   bInheritHandle <>;
} SECURITY_ATTRIBUTES, *PSECURITY_ATTRIBUTES;
```

- **bInheritHandle** Specifies whether the returned handle is inherited when a new process is created. If this member is TRUE, the new process inherits the handle.

lpSecurityDescriptor

- Pointer to a security descriptor for the object that controls the sharing of it. If NULL is specified for this member, the object is assigned the default security descriptor of the calling process. This is not the same as granting access to everyone by assigning a null DACL. The default security descriptor is based on the default DACL of the access token belonging to the calling process. By default, the default DACL in the access token of a process allows access only to the user represented by the access token. If other users must access the object, you can either create a security descriptor with a null DACL, or add ACEs to the DACL that grants access to a group of users.

SECURITY_DESCRIPTOR

- The **SECURITY_DESCRIPTOR** structure contains the security information associated with an object. Applications use this structure to set and query an object's security status.

```
typedef struct _SECURITY_DESCRIPTOR {
  BYTE  Revision;
  BYTE  Sbz1;
  SECURITY_DESCRIPTOR_CONTROL Control;
  PSID Owner;
  PSID Group;
  PACL Sacl;
  PACL Dacl;
} SECURITY_DESCRIPTOR, *PISECURITY_DESCRIPTOR;
typedef PVOID PSECURITY_DESCRIPTOR;
```

Object security components

- A security descriptor includes information that specifies the following components of an object's security:
  - An owner (SID)
  - A primary group (SID)
  - A discretionary ACL
  - A system ACL
  - Qualifiers for the preceding items
- Security descriptors use access-control lists (ACLs) and security identifiers (SIDs) to specify the information in this list.
- A security descriptor can be in absolute or self-relative form. In self-relative form, all members of the structure are located contiguously in memory. In absolute form, the structure only contains pointers to the members.

Secure Object Creation

- The **CreateWindowStation** function creates a window station object. It retrieves a handle that can be used to access the window station. A window station is a secure object that contains a set of global atoms, a clipboard, and a set of desktop objects.

```
HWINSTA CreateWindowStation(
  LPTSTR lpwinsta <>,                // new window station name
  DWORD dwReserved <>,               // reserved; must be zero
  ACCESS_MASK dwDesiredAccess <>,    // requested access
  LPSECURITY_ATTRIBUTES lpsa <>      // security attributes
```

IServerSecurity::QueryBlanket

- Called by the server to find out about the client that invoked one of its methods.

```
HRESULT QueryBlanket(
  DWORD *  pAuthnSvc <>,      //Pointer to the current authentication
                            // service
  DWORD *  pAuthzSvc <>,      //Pointer to the current authorization
                            // service
  OLECHAR **  pServerPrincName <>,
                            //Pointer to the current principal name
  DWORD * pAuthnLevel <>,     //Pointer to the current authentication level
  DWORD * pImpLevel <>,       //Reserved for future use; must be NULL
  RPC_AUTHZ_HANDLE *  pPrivs <>,
                            //Pointer to handle to privilege information
  DWORD *  pCapabilities <>  //Pointer to flags indicating further
                            // capabilities
);
```

CoGetCallContext

- Retrieves the context of the current call on the current thread.
- **HRESULT CoGetCallContext(**
- **REFIID** *riid <>*, //Interface identifier
- **void **** *ppInterface <>* //Address of output variable that receives the
- // interface pointer requested in *riid*
- **);**
- **Parameters**
- *riid*
- [in] Interface identifier (IID) of the call context that is being requested. If you are using the default call context supported by standard marshaling, **IID_IServerSecurity** is available. For COM+ applications using role-based security, **IID_ISecurityCallContext** is available.
- *ppInterface*
- [out] Address of pointer variable that receives the interface pointer requested in *riid*. Upon successful return, **ppInterface* contains the requested interface pointer.

CoQueryClientBlanket

- Called by the server to find out about the client that invoked the method executing on the current thread.

```
HRESULT CoQueryClientBlanket(
  DWORD* pAuthnSvc <>, //Pointer to the current authentication service
  DWORD* pAuthzSvc <>, //Pointer to the current authorization service
  OLECHAR ** pServerPrincName <>,
                       //Pointer to the current principal name
  DWORD * pAuthnLevel <>,
                       //Pointer to the current authentication level
  DWORD * pImpLevel <>,
                       //Reserved for future use; must be NULL
  RPC_AUTHZ_HANDLE * pPrivs <>,
                       //Pointer to handle to privilege information
  DWORD ** pCapabilities <>
                       //Pointer to flags indicating further
                       // capabilities of the proxy
```

IServerSecurity::ImpersonateClient

- Allows a server to impersonate a client for the duration of a call.
- **`HRESULT ImpersonateClient( )`**
- Normally, a method executes on a thread that uses the access token of the process. However, when impersonating a client, the server runs in the client's security context so that the server has access to the resources that the client has access to.
- When impersonation is necessary, the server calls the **ImpersonateClient** method to cause an access token representing the client's credentials to be assigned to the current thread. This thread token is used for access checks. **RevertToSelf** restores the current thread's access token.

Impersonation level

- What the server can do on behalf of the client depends on the impersonation level set by the client, which is specified using one of the RPC_C_IMP_LEVEL_xxx constants.
- The server may impersonate the client on a secure call at identify, impersonate, or delegate level.
- The identity presented to a server called during impersonation depends on the type of cloaking value, if any, that is set by the client.
- Traditionally, impersonation information is not nested - the last call to any Win32 impersonation mechanism overrides any previous impersonation.

LookupAccountName

- The **LookupAccountName** function accepts the name of a system and an account as input. It retrieves a security identifier (SID) for the account and the name of the domain on which the account was found.

```
BOOL LookupAccountName(
  LPCTSTR lpSystemName <>,    // system name
  LPCTSTR lpAccountName <>,   // account name
  PSID Sid <>,                // security identifier
  LPDWORD cbSid <>,           // size of security identifier
  LPTSTR DomainName <>,       // domain name
  LPDWORD cbDomainName <>,    // size of domain name
  PSID_NAME_USE peUse <>      // SID-type indicator
);
```

LookupAccountSid

- The **LookupAccountSid** function accepts a security identifier (SID) as input. It retrieves the name of the account for this SID and the name of the first domain on which this SID is found.

```
BOOL LookupAccountSid(
  LPCTSTR lpSystemName <>,  // name of local or remote computer
  PSID Sid <>,              // security identifier
  LPTSTR Name <>,           // account name buffer
  LPDWORD cbName <>,        // size of account name buffer
  LPTSTR DomainName <>,     // domain name
  LPDWORD cbDomainName <>,  // size of domain name buffer
  PSID_NAME_USE peUse <>    // SID type
);
```

ISecurityCallContext

- The **ISecurityCallContext** interface provides access to security methods and information about the security call context of the current call.
- COM+ applications that use role-based security have access to the security call context property collection through this interface.
- You can obtain information about any caller in the chain of callers, as well as methods specific to COM+ role-based security.

ISecurityCallContext

- Call methods of this interface to gain access to items in the security call context collection. The following items are available:
  - Number of Callers
  - Minimum Authentication Level
  - Callers
  - Direct Caller
  - Original Caller

ISecurityCallContext

- You can also use this interface to find out whether the direct caller or the specified user is in a particular role, such as **IsCallerInRole** or **IsUserInRole**, and whether role-based security is enabled for the object (**IsSecurityEnabled**).
- **ISecurityCallContext MethodsDescription**
- get_Count Retrieves the number of properties in the security context collection.
- Get_Item Retrieves a specified property in the security call context collection.
- Get__NewEnum Retrieves an iterator for the security call context collection.
- IsCallerInRole Determines whether the direct caller is in the specified role.
- IsSecurityEnabled Determines whether security is enabled for the object.
- IsUserInRole Determines whether the specified user is in the specified role.

Summary and Conclusion

- A “Wintel” shop has significant configuration tools for enhancing application security.
- The risk management document of the MSF process should address the configuration of security in the application.