MQSeries Security White Paper

Revision Date: October 23, 1998

Stuart C Jones
MQSeries Technical Strategy
IBM Hursley

stuartc_jones@uk.ibm.com

Table of Contents

Introduction

The purpose of this document is to explain security in an MQSeries environment and to give some indications of the directions for the MQSeries product family in the security arena. This document is divided into 4 sections.

1. Access control and the way that the various MQSeries resources are protected
2. Security for distributed messaging, including the different ways that the security facilities can be customized.
3. Message level security - in terms of the (limited) existing facilities and the vendor products available
4. Future directions for security in an MQSeries environment.

Note that it is not the intention to significantly duplicate information that is provided in the existing MQSeries manuals. Thus, where appropriate, references are included to the appropriate MQSeries documentation.

Platform Coverage

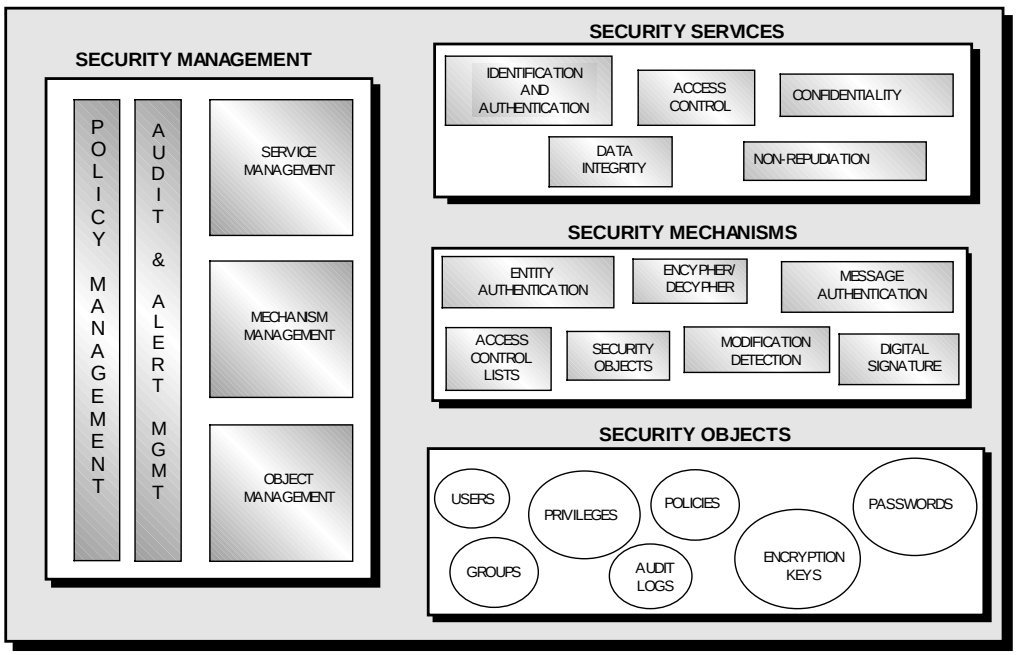
The facilities described in the document are not applicable to all of the MQSeries product family. At the time of writing, these facilities are **not** available on the following products:

- MQSeries for Windows
- MQSeries for unixware
- MQSeries for VSE/ESA
- MQSeries for SCO UNIX

There has been an announcement that updated versions of MQSeries will be made available for unixware and SCO UNIX. Thus, these facilities will be provided for these environments in the near future (at the time of writing).

The Security Environment

The following diagram illustrates the set of security facilities that *might* be available in a secured environment. Not all of these facilities *need* to be available on each node, though many will be.



This diagram also shows the *relative* positioning of the various security facilities. For example, security services make use of security mechanisms which, in turn, act on security objects. The focus for MQSeries security is with security services and security management. MQSeries needs to provide facilities to support the various security services, though does not implement security mechanisms. Also, MQSeries needs to implement some security management functions, namely security administration and auditing.

It is important to note that MQSeries does not get involved with identification and authentication of users. So, users do not sign on to MQSeries. It is assumed that users will have identified themselves to 'the system' and that 'the system' will have authenticated the user before MQSeries facilities are used. But, MQSeries will propagate the userid in the header of each message that is put to a queue. Thus, identification services are provided in this way.

MQSeries Access Control

Introduction

This section is concerned with access control and the way that MQSeries protects the resources that it owns. There is a description of what access control measures *might* be implemented and what the queue mangers *do implement* for each platform. This includes a summary of the principle differences between the implementations. There is also a walkthrough of the typical path through the access control functions provided by MQSeries.

Overview

(Probably) the primary purpose of security is to protect resources from unauthorized access. A major part of this protection is checking whether a user of a system is authorized to access a particular resource. There are a number of places where this authorization may be checked:

- Physical checks. Even though the issue under discussion is the protection of (primarily) software resources, it is possible to physically protect resources. This is generally achieved by allowing only authorized to users to physically use a machine – either by controlling access to a building or room in which a system is housed.
- Software checks. There are several points at which the software can control access to resources.
    - Access to the system as a whole; most environments require that users sign on to a system in some way before any of it's components may be accessed.
    - Access to system definition functions. If a component cannot be installed or configured then it certainly cannot be used.
    - Access to link libraries. It is not possible to access a component programmatically if the required API cannot be either compiled or linked.
    - When programs try to access a component, it is possible to control those users who are to be granted access.
    - When programs access individual resources, it is possible to control those users who are to be granted access.

The above 'checkpoints' are, clearly, hierarchical. It is possible to implement none, one, many or all of these facilities, as required by the installation. It is the responsibility of each enterprise to decide which of these facilities are required, dependant upon the value that they place on the resources to be protected and the level to which they consider that those resources are exposed. Another consideration will be (should be) the amount of resource (in financial and human terms) that an enterprise is willing to spend on this protection…these facilities rarely come for free.

Before any checking can be performed, it is clearly important that there be 'someone' to check up on. Thus it is a requirement of access control that the 'someone' who requires access to resources is identified in some way.

MQSeries 'Checkpoints'

MQSeries (and the operating system) provides facilities to control access to MQSeries resources. Clearly, neither the operating system nor MQSeries can provide the physical 'checkpoints' described above. If these are to be provided, then they are outside the scope of any software components. In terms of software checks, the following is provided:

- The operating system can control access to program libraries and, therefore, control access to system installation/configuration programs (such as *crtmqm*, *strmqm*, *runmqsc* and the MVS batch jobs which set up a system on MVS/ESA) and to the MQSeries API link libraries.
- MQSeries provides mechanisms to limit access to its own components, as follows:
  - MQSeries utilities for the non-MVS queue managers (such as *crtmqm*, *strmqm* and *runmqsc*) may only be used by suitably authorized users.
  - MQSeries controls which users may connect to the Queue Manager (via MQCONN or MQCONNX)
  - MQSeries controls which users may access MQSeries resources and in what manner those resources may be accessed. The resources which may be controlled are:
    - Queue manager object
    - Queues
    - Namelists (MVS only)
    - Processes
    - Channels (MVS only)
      Note that, for the majority of queue managers, channels are **not** MQSeries objects and so access to the channels may not be controlled.

MQSeries Access Control Implementation

MQSeries provides access control facilities in two different ways, depending upon the platform. The MVS platform has one implementation and *all* other platforms have a separate implementation, common to all of those platforms.

MQSeries User Identifiers

In order to control access to MQSeries resources, the queue manager needs to be able to identify the user who is requesting access to a particular resource. This identification is done when an MQSeries application first connects to the queue manager, when MQCONN (or MQCONNX) is used. The queue manager queries the application environment to discover which user is associated with the connecting application. The user identifier returned to MQSeries is that understood by the underlying environment and may not (necessarily) be the identifier that might be expected. For example, some environments will return multiple identifiers (see MVS below) and identifiers that the underlying environment does not recognize will not be returned – such as DCE principals, userids in some MVS environments, etc.

As noted above, the identifier returned to MQSeries is dependent upon the application environment. The user identifier returned to MQSeries for different environments is as follows:

- CICS and IMS on MVS
  MVS address space userid **and** user identifier associated with the (CICS or IMS) transaction.
- MVS Mover
  The MVS Mover works in a similar manner to CICS and IMS in that 2 userids may be used for access control checks. The address space userid and the userid associated with the message channel agent may be used. For more details, see the section below on distributed messaging.
- MVS TSO
  TSO userid
- MVS batch/TSO
  MVS address space userid
  Note that all MVS address spaces other than those named above are treated as MVS batch address spaces.
- UNIX queue managers
  User name associated with the connecting process
  A UNIX application may have a 'preset' user name, known as an *effective* user name. This preset characteristic will be in addition to the user name associated with the actual user of the application, known as the *real* user name. MQSeries will always use the **real** user name for any access control checks that need to be made.
- All other queue managers
  Userid associated with the connecting process
- MQSeries and non-MVS transaction monitors
  If the application is running in a transaction monitor environment, then the connection between the application and the queue manager is via the X/Open XA interface. This interface does not provide a mechanism for passing a user identifier to the queue manager and so MQSeries assumes a default user name.

User Identifiers and Groups

Most operating environments support the notion of user identifiers belonging to groups. Depending upon the specific environment the way that MQSeries works with user identifiers and groups differs:

- MVS supports both userids and groups. When MQSeries performs access control checks it passes a userid to the external security manager
- UNIX systems support group based security. MQSeries access control lists are based upon groups *only*. The groups that a particular user name belongs to are queried and each group is tested to check its access to the required resource. Only one group (not all) needs to have access to the required resource.
  When a user name is granted access to a particular resource, the user name's *primary group* is included in the MQSeries access control list and *not* the user name.
- OpenVMS supports the concepts of accounts, users, groups and identifiers! These concepts are related as follows:

- An account is represented by *[group number, user number]* (values from 1-777). Thus, account Fred may be represented as [123,005]. An account may be associated with *only one* group.
- Accounts may be granted one or many identifiers and a single identifier may be associated with many accounts. This is similar to the UNIX concept of groups, though there is no *primary* identifier for any particular account.

MQSeries controls access to resources based upon accounts and identifiers.

- Windows NT systems support userids and groups. MQSeries access control lists are based on both userids and groups. Access control checks are the same as for UNIX except that individual userids may appear in the access control list as well.
- Tandem NSK support user names and group names.

MQSeries recognizes only the first 12 characters of any userid. This is sufficient for most environments but does not satisfy requirements in all. There is more detail on this subject in the section on 'Platform Level Issues' below.

MVS Access Control Facilities

MVS provides access control facilities via the SAF (System Access Facility) interface. This is the standard interface used to enable MVS subsystems to provide security facilities. All of the common external security manager products (both IBM and non-IBM) conform to the SAF interface and so any of these products may be used with MQSeries. The list of currently supported products includes RACF, Top Secret and ACF2...the latter two products being provided by Computer Associates. These products are usually referred to as External Security Managers, ESMs.
There are 3 main components to the MVS security facilities – security switches, access control profiles and other facilities.

1. MQSeries implements security switches (i.e. whether or not to use various access control features) by the use of *Switch Profiles*. Switch Profiles are regular security manager profiles and it is the way that the profiles are treated that distinguishes them from other access control profiles. MQSeries will check for the ***existence*** of these switch profiles – rather than the ***access*** that a particular userid has to the profile (which is the standard use of such profiles). Thus, it is possible to control the type of checking that MQSeries will perform by creating the appropriate switch profiles. There are a number of switch profiles, allowing very granular control of just which MQSeries resource types are checked. (The complete list of switch profiles is contained in the MQSeries System Management Guide).
   There is a further security switch…the *RESSEC Profile*. This profile controls the ***number*** of userids that will be used for any particular access control check. Depending upon the access that an address space (MQSeries adapter) userid has to the RESSEC profile, there may be 0,1 or 2 access control checks made when an attempt is made to access an MQSeries resource. Note that inappropriate access to the RESSEC profile may cause either an excessive amount of resource checking or not enough!
2. MQSeries defines a set of ESM profiles that are used to restrict access to resources and to commands. Userids are granted access to these profiles as appropriate and the

ESM (on behalf of MQSeries) will verify this access when access control checks are made. The different level of access required to these profiles for each different style of MQSeries activity is documented in the MQSeries System Management Guide.

3. MQSeries provides a set of (MQSC) commands to control how the above profiles and userids are used within the queue manager. Because of the expense of continually accessing the above profiles and the capabilities of each user within an external security manager, this information is cached. The commands enable the cache to be completely or selectively refreshed.

Note that all of the profiles referenced above, the switch profiles and the access control profiles are owned and managed by the ESM and not by MQSeries.

Non-MVS Access Control Facilities

For the non-MVS systems, there is no common mechanism such as MVS SAF (across all of the platforms) for provision of access control facilities. Consequently, MQSeries provides its own set of facilities, as follows:

- An access control interface is provided, as an instance of an MQSeries Installable Service. MQSeries provides a documented interface (called the Authorization Service) to an access control component. This interface is documented in the MQSeries Programmable Systems Management Guide.
- An implementation of an access control manager, which conforms to the Authorization Service interface above. This implementation is known as the **O**bject **A**uthority **M**anager, or OAM. The OAM maintains an access control list for each resource that it controls.
  The OAM is passed a principal, a resource name and an access type request. It will then either grant or reject access based upon the access control lists that it maintains. This mechanism becomes significant when operating system differences are considered below.
  Note that the OAM may be replaced by any user or vendor written component which conforms to the Authorization Service interface.
- A set of utility functions to administer the access control lists used by the OAM. These utility functions are *setmqaut*, used to define access capabilities and *dspmqaut*, used to display access capabilities. These utilities are documented in the appropriate MQSeries System Administration manual.
  These utilities interact with the OAM which means that the queue manager must be active in order to use the utilities.

Differences Between The MVS & Non-MVS Access Control Mechanisms

There are several differences between the MVS and non-MVS implementations, as follows:

- The MVS queue manager provides more granularity for *how much* access control checking is performed – via the switch profiles. The non-MVS queue managers do not provide any such granularity…security checking is either active or inactive.
- The level of access control granularity provided differs. In some areas, MVS provides more granular function than the other platforms, particularly for control of MQSeries administration capabilities. In other areas, the non-MVS queue managers provide the more granular functions - for instance, in differentiating between get and put capabilities for a queue.
  Note that **neither** implementation provides a subset of the other.
- MVS provides the ability to have generic access control profiles, spanning many individual resources (owned by the same queue manager) with a single profile. The non-MVS queue managers do not support generic profiles for access control functions. The primary reason for this is that while the MVS implementation provides a single set of profiles, the non-MVS implementation provides for a separate ACL for *each* resource. While this does not implicitly prevent the OAM supporting generic profiles, they have not been implemented.
- There is a separate instance of the OAM for each queue manager whereas the MVS ESM may provide services to any number of queue managers on the same MVS image. This means that there is a separate set of ACLs for each queue manager. Further, each ACL must be maintained separately (unless some management tool is used which allows a central point of configuration function). The Tivoli product set supports this function.
- The MVS ACLs are not directly associated with the MQSeries objects. The first consequence of this is that MVS ACLs may be maintained without the queue manager being active. The second consequence is that an MVS ACL may be created without the associated object existing.
- It is not possible to refresh the access control cache or userid cache for the non-MVS queue managers. If access control permissions or userid capabilities are changed then the queue manager must be re-cycled.

Alternate Userids

As described above, the user identifier which the queue manager uses for access control checks is, generally, a userid obtained from the operating environment. When an application issues an MQCONN, the queue manager queries the operating environment for the userid associated with this application. This userid is retained and used for access control checks. There is an exception to this mechanism; a **suitably authorized** application may issue an MQOPEN (or MQPUT1) specifying an *alternate* userid. Thus, USER1 may issue an MQOPEN specifying USER2 as an alternate userid and any access control checks will be made on USER2, not USER1.
Future use of this alternate userid is then platform specific:

- For non-MVS systems, the alternate userid is used *only* for access control checks on the object being opened. Specifying an alternate userid **does not** change the userid associated with an application. Thus, for the above example, any messages put to a queue opened using USER2 will (by default) contain USER1 in the MQMD context fields (explained below).
- For MVS systems, the alternate userid is used in subsequent operations for that object handle. Using the above example again, if a queue is opened with alternate userid USER2 then any MQPUT operations will have MQMD.UserIdentifier set to USER2.

The authorization to use an alternate userid is specified via the appropriate access control mechanism (RACF profile, setmqaut, etc). Note that, for MVS, USER1 requires specific permission to use USER2. For non-MVS platforms, permission to use *any* alternate userid is all that is required.

Access Control Walkthrough

Given the above mechanisms for controlling access to MQSeries resources, it is useful to walk through the process that occurs for a typical application trying to access MQSeries resources. The description below assumes that, for MVS, all access control checking is active:

- MQCONN/MQCONNX
  This is the first point at which an application will be associated with a particular queue manager. As a part of connect processing the queue manager interrogates the operating environment to discover the user identifier associated with the application. This may be a query to the operating system or to a transaction manager such as CICS or IMS. MQSeries will verify that the user identifier returned by the operating environment is authorized to connect to the queue manager. The user identifier is then retained for any future access control checks.
  The user identifier which is returned will be the one understood by the base environment. This means that – today – MQSeries will not recognize a DCE principal or any other identifier that is not recognized by the base system.
- MQOPEN/MQPUT1
  MQSeries objects are accessed by opening the resource and then issuing commands against the resource. This is true whether the MQOPEN/MQPUT1 is explicitly or implicitly performed. **All** resource checks are performed at the time the object is opened, using MQOPEN or MQPUT1, rather than when the resource is actually accessed. It is for this reason that the MQOPEN (or MQPUT1) must specify the type of access that is requested (i.e. GET, PUT, BROWSE, INQUIRE, etc.).
  The resource for which access permission is checked *is the resource named on the MQ API command*. If this resource is an Alias or Remote queue definition, the resolved resource is **not** checked. This means that a user does not require access to a resolved queue such as a transmission queue. A further implication of this is that MQSeries administrators should restrict those users who are permitted to define (particularly) alias or remote queues.
  If the object being opened is explicitly referenced such that both the queue and queue manager names are provided then the transmission queue associated with the target queue manager is the resource for which access is checked.

- MQPUT/MQGET
  No access control checks are performed.
- MQCLOSE
  If an MQCLOSE will result in a dynamic queue being deleted then there will be a check that the user identifier is suitably authorized to delete the queue. This is the only exception to the rule that all API access control checks are performed at MQOPEN/PUT1

The access control permissions required for each type of access to MQSeries objects are documented in the appropriate Systems Administration Guide for the MQSeries platform in question.

Platform Level Issues

There are a number of platform specific exceptions to the above text, as follows:

- The Tandem NSK environment has two different levels of security support, depending upon the software available on the particular NSK system.
  - If the base level of security support is available then a Tandem user is represented as '*group-name.user-name*' … the user and the group to which that user belongs. Both the group-name and the user-name are up to 8 (case insensitive) characters in length and a user may be associated with only one group. Clearly, it is not possible to accommodate both of these values in the 12 character MQMD field. So, MQSeries recognizes the group-name *only* for security processing, the user-name is ignored.
    This means that MQSeries for Tandem NSK does not operate seamlessly in a heterogeneous environment as messages from different users (in the same Tandem group) will have the same group-name in MQMD.UserIdentifier and so it will not be possible to distinguish which user put the message.
  - If the Tandem SAFEGUARD product is installed then a particular user may be a member of more than one group and the user may be represented by a (case sensitive) 32 character Alias.
    When SAFEGUARD is used, MQSeries is not able to take advantage of any of these advanced features.

  Both of the above issues, support of heterogeneous operations and support of Tandem SAFEGUARD will be addressed in a future release of the MQSeries for Tandem NSK product.

- The Windows NT platform has two significant differences from many other operating systems with respect to security:
  1. Windows NT user identifiers are up to 20 characters in length and may contain embedded blanks.
  2. Windows NT user identifier may be domain qualified – the same userid may exist in more than one NT security domain and the each instance of the userid is distinct, allowing differing access control permissions.

While it is likely that NT V5 will ease the issue of domain qualified userids, there is still the issue of the format of the userid itself. Currently, MQSeries will recognize up to 12 characters of an NT userid and any userid will be 'blank delimited' (and so may not contain embedded blanks). In many instances this is an acceptable limitation as the primary use of MQSeries is *heterogeneous* processing and most other operating systems do not recognize blank embedded or long userids. However, this is a limitation which cannot be allowed to continue, particularly for NT only environments, and will be corrected in a future MQSeries release.

MQSeries Message Context Fields

When a message is put on a queue, there is security context information associated with the message. This context information in placed in the MQ Message Descriptor, the MQMD. There are 8 fields, divided into two sections, as follows:

1. Identity Context
   - UserIdentifier
     This is a 1-12 character field which contains the userid associated with the putting application. For userids which are longer than 12 characters there are platforms specific approaches, documented above.
   - Accounting Token
     This field is used to identify the *instance* of the application which has generated the message. If not set, the queue manager will set an environment dependent value which is documented in the MQSeries Application Programming Reference.
   - ApplIdentityData
     This is application dependant data and MQSeries does not define its format or place a default value in it. It may only be used by suitably authorized applications.
2. Origin Context
   - PutApplType
     This is the type of application which has PUT the message and identifies the environment in which the application was running, such as CICS, IMS, AIX,...
   - PutApplName
     This is the name of the application which has PUT the message and will be the first 28 bytes of the fully qualified pathname for most environments and a transaction ID where appropriate.
   - PutDate
     The format is YYYYMMDD, GMT. Note that MQSeries date formats are year 2000 ready.
   - PutTime
     The format is HHMMSSTH, GMT
   - ApplOriginData
     This is application dependant data and MQSeries does not define its format or place a default value in it. It may only be used by suitably authorized applications.

The MQMD is passed to the queue manager by the application. Thus the MQMD is an input parameter for MQPUT/MQPUT1. However, the context section of the MQMD may **only** be populated by the application if that application is suitably authorized so there is no security exposure associated with having the context fields as a part of the MQMD. An application must open the target queue with the appropriate options for modifying the context fields and put messages using the appropriate options. This leads to 3 possible failure scenarios:

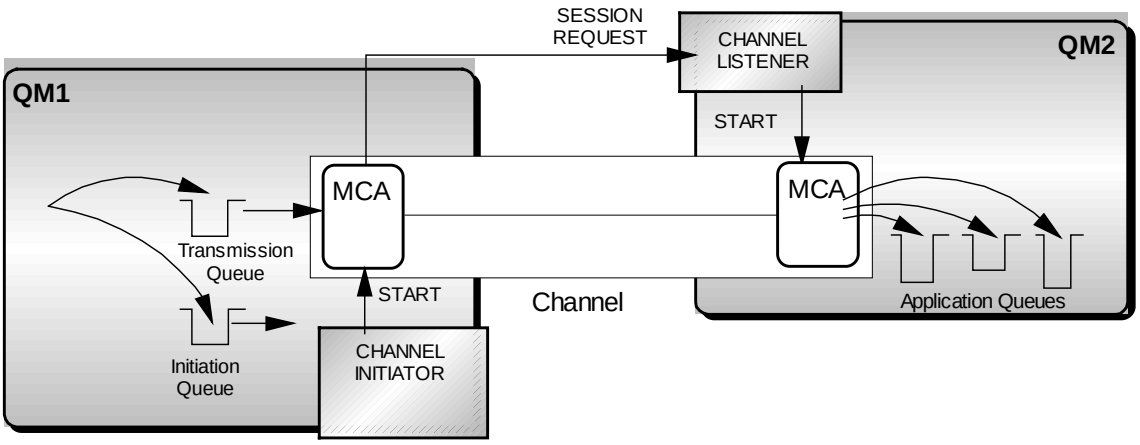
1. If an application attempts to open a queue using the appropriate options and is not suitably authorized, **the open will fail** with return code MQRC_NOT_AUTHORIZED.
2. If the queue is opened without specifying the appropriate options and attempts to populate the context fields without specifying the appropriate MQPUT options, **the values in the MQMD context fields are ignored**.
3. If the queue is opened without specifying the appropriate options and attempts to populate the context fields specifying the appropriate MQPUT options, **the MQPUT fails** because the options for MQOPEN and MQPUT are inconsistent.

The mechanism for controlling access to the MQMD context fields is fully documented in the MQSeries Application Programming Guide.

Distributed Messaging

Introduction

The MQSeries distributed messaging component may be represented by the following diagram:



This component is responsible for passing messages to partner queue managers and receiving messages from partner queue managers. It is made up of the following components:

- Transmission Queue
  The transmission queue (also known as an XmitQ) is the local queue on which messages are safe-stored before being passed to a partner queue manager. An XmitQ is just like a regular MQSeries local queue except for the USAGE parameter, which is set to XMITQ.
- Message Channel Agent
  The MCAs provide the functions which reliably exchange messages with other queue managers and which provide the linkage between MQI clients and servers. A pair of MCAs – one in the sending queue manager and one in the receiving queue manager - is known as a channel.
  The MCA which initiates the channel is known as the *caller* MCA. The partner MCA is known as the *responder* MCA.
  MCAs are (privileged) MQ applications and are subject to the same resource access controls as any other application.
- Channel Initiator
  This component is responsible for starting caller MCAs when the queue manager administrator has configured the channel for automatic initialization. (This is done by defining the transmission queue as a triggered queue).
- Listener
  When MCAs connect to one another, it is advantageous (required, really) to have a component on the 'receiving' queue manager which provides automatic initialization

of the responder MCA. This allows the message exchange process to be automated. In some cases, this listener is a part of the underlying transport mechanism, such as TCP/IP's inetd (a transport listener). In other cases, it is provided by the queue manager (an MQSeries listener).

On MVS/ESA, the distributed messaging component is usually provided by an address space known as the MVS Mover (or CHINIT…for channel initiator). This is a separate address space from the queue manager address space and implements all of the above components (except for the transmission queues which are a part of the queue manager). It is also possible to provide distributed messaging via a set of CICS/ESA transactions, though this is rarely used today and is not the recommended component. These CICS transactions will not be discussed further. On the non-MVS queue mangers, the above components are all provided as separate entities (i.e. processes or threads).

The distributed messaging component provides a minimal set of security features. A further set of security features is enabled via the MQSeries exits.

Base Channel Security Facilities

MCA Userids

The MCAs are simply specialized MQSeries applications. As such, they access MQSeries resources and require a user identifier in order to enable such access control to be enabled. As with other applications, the user identifier associated with an MCA is environment dependent, as follows:

- Caller MCAs
  These MCAs are started either as individual processes, as a thread of the (non-MVS) channel initiator or as a dispatchable unit of the MVS Mover. The user identifier associated with the caller MCA is the user identifier associated either with the parent process (for threads) or with the process causing the MCA to be started.
  In general, this userid requires access to the queue manager, the XmitQ, the DLQ and any resources which may be accessed by channel exits, which are discussed below.
- Responder MCAs
  These MCAs are started as a result of a request from a caller MCA. Like caller MCAs, responders may be started either as individual processes, as threads of the (non-MVS) MQSeries listener or as a dispatchable unit of the MVS Mover. There are two additional considerations for responder MCAs:
  - The caller MCA may indicate the userid to be used for the responder (known as the network userid as it arrives from the partner MCA over the network). This is available if the transport is APPC, conversation level security is being used and the responder MCA is not started as a (non-MVS) thread.
  - The configuration of the responder MCA may specify the user identifier that MQSeries is to use for this responder MCA. The MCAUSER parameter of the Channel definition is used.

  Thus the user identifier associated with a responder MCA may be any of the user identifier associated with the parent process (for threads), with the process causing

the MCA to be started or with the caller MCA, via the network userid. The preference order used is network userid, MCAUSER and finally the owning/starting process userid.

MVS provides a further extension to this by associating **two** userids with responder MCAs. The first userid is intended to be for the network userid and the second for the MCAUSER. In both cases, the process userid (userid associated with MVS Mover address space) is used if network userid or MCAUSER is not available. When access control checks are performed for responder MCAs, there are then 2 checks to perform…one for each userid. Provision of two userids may make setup of MCA within the MVS Mover quite complex. The way that this setup should be handled is documented in the security section of the MQSeries for MVS System Management Guide.

Placing Messages on Target Queues

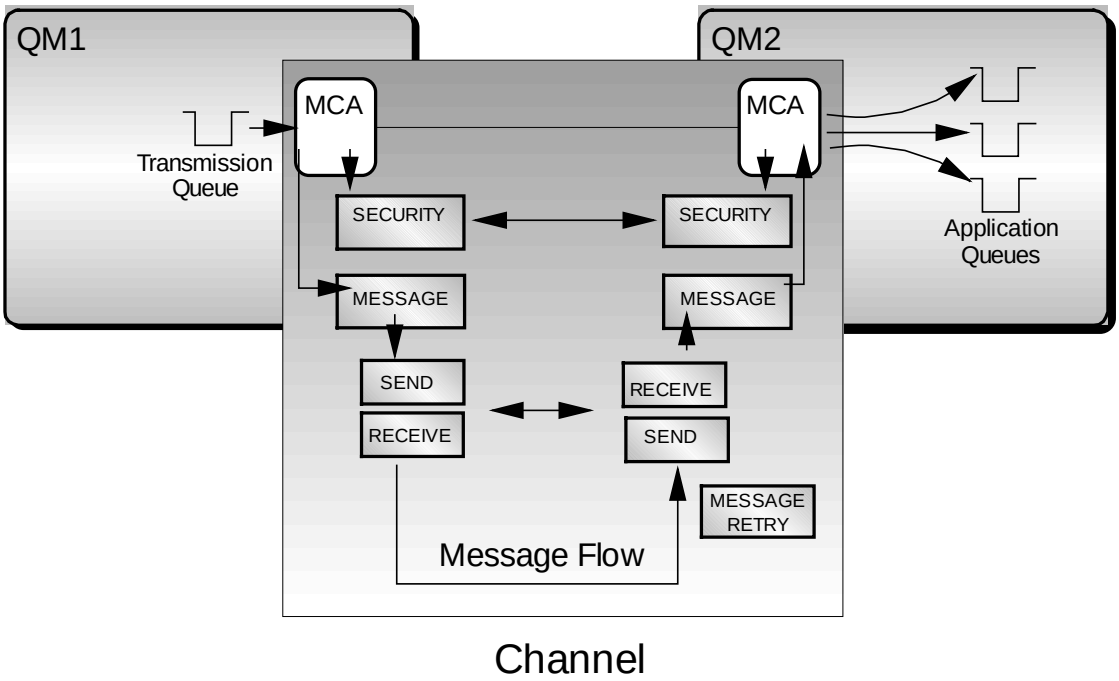
When an MCA puts messages on a target queue, it is necessary for the receiving MCA to open the target queue and put messages on it. The MCA uses the MQSeries API and, therefore, may involve access control checks. There is a choice of userids available for this authorization check; there is a userid (two for MVS) associated with the MCA and there is a userid associated with the incoming message – the UserIdentifier from the MQMD context fields. The choice of which userid to use is a configuration option for each channel – the PUTAUT parameter (for PUT authorization) of the Channel definition.
The choice made may have a significant effect on the number of user identifiers that a receiving system needs to be aware of.

- If PUTAUT(DEFAULT) is selected then the userid associated with the receiving MCA is used. This will be a userid that is *already defined* on the local system.
- If PUTAUT(CONTEXT) is selected then the userid associated with the message is used. This will (very likely) be a userid from a remote system which must be recognized by the local system. The consequence of this is that (possible very many) userids from remote systems need to be recognized (and, therefore, defined) on the local system. This may be a significant management overhead.

For this reason, receiving MCAs are often configured to use PUTAUT(DEFAULT), though this is certainly not a requirement.

MQSeries Channel Exits

MQSeries channels provide a set of exit points enabling a channel to be customized, as follows:



Although the exits are not specifically oriented toward security, a number of the exits can provide security enhancements for MQSeries channels, as follows:

- Security exit
  The security exit is intended, primarily, for (mutual) authentication of partner MCAs when they connect to one another. This exit is invoked after the MCAs have connected but *before* any user data is exchanged. The exits are (almost) invariably implemented as a complementary pair – one for each MCA, though this is not an absolute requirement. The exits may have an **arbitrary** exchange of data (from the viewpoint of the MCAs) in order to verify the identity of the partner MCA. Once the exits are satisfied that the partners are genuine, message data may be exchanged between the MCAs.
  The naming of this exit is misleading as the implication is that it is *only* for security and that **all** security functions need to be performed within it. Both of these are incorrect. Because the data exchange is arbitrary, any exchange is possible and so functions unconnected with security may be implemented. (MQSeries Supportpac MS05 is a good example of a non-security function). Also the exit does not have any access to message data and so cannot enable any message level security functions such as encryption.
  So, the primary (security) function of a security exit in the server/server environment is MCA authentication. There is a further (security) function in the client/server environment and this is discussed below.
- Message exit

The message exit is called once for each message (in each MCA) and has access to both the MQSeries message headers and the message data. This exit is useful for functions which require access to all of the message data, rather than a portion of it. Some enterprises choose to use the message exit for signing messages, adding data integrity tokens or for encrypting data.

- Send/Receive exit
  Because MQSeries allows message sizes greater than the capacity of the underlying transport, it is sometimes necessary for the MCA to segment messages before sending to the partner MCA – which will reconstitute them at the receiving side. For each segment, the send exit is called at the sending MCA and the receive exit called at the receiving MCA.
  Usually, the send/receive exits are used where the structure of the message data is not significant. In fact, MQSeries does not expose the data content of each segment so it is important to treat each segment as an opaque 'blob'. But, it is quite possible to implement security functions such as signing messages, adding integrity tokens and encryption within the send/receive exits if required.

- The Message Retry Exit is not applicable to security functions and so is not discussed here.

There are a few points to note regarding the usage of these exits:

- Although the exits are represented as pairs in the above diagram, there is no absolute requirement that exits be implemented in this way. It is possible, for instance, to encrypt a message in a sending message exit and to decrypt the message in an application.
- If the message exit is used for encryption, the message headers (MQXQH, MQMD) should **not** be encrypted. This is because data conversion for the message headers takes place either after the sending message exit is called or before the receiving message exit. If the headers are encrypted then the conversion will fail and the channel will stop.
  MQSeries message header conversion takes place at this point in the processing because the message exit is permitted to modify the header data.
- In general, the channel exits are user or vendor supplied. MQSeries V5 is an exception in that security, message and send/receive exits are supplied to provide authentication and encryption services. These exits require that a DCE environment is present.

Additional MQI Client/Server Considerations

The MQI client/server environment uses MCAs and channel exits for communication between the client and the server. For this environment, there are some additional security considerations with respect to the exits.

The purpose of the client/server component of MQSeries is to package and transport MQ API requests and responses between the client application and the SVCONN MCA which

is acting as a proxy on behalf of the client. For some environments, it is important that the security context (the user identifier) of the client is transferred to the proxy as well. In this way, any commands executed by the proxy will use the authority associated with the client userid and not the default userid of the proxy. There are three ways of achieving this with today's queue managers:

1. Define a unique SVRCONN channel for *each* client and specify the MCAUSER parameter in the Channel definition to be that of the client. While this is certainly possible, it is not a particularly workable (scalable) solution, particularly from the systems management viewpoint.
2. Each client has an environment variable, MQ_USER_ID, available. If this environment variable exists then it will be passed to the SVRCONN MCA and may be used as the userid of the proxy **only if** the SVRCONN MCAUSER is blank. The consequence of this last statement is that the server has control of whether or not the client userid environment variable is used.
   Use of the environment variables for security (authentication) purposes is not recommended, as they may be set by any user (authorized or not). The environment variables will be removed in a future release of the queue manager.
   Currently, the default value of MCAUSER, for a SVRCONN channel, varies by platform and release level. For MQSeries V2, the default usually is a low privilege user such as 'nobody' on UNIX. For MQSeries V5 and MQSeries on MVS, the default value is blanks. Now, while this make for easier connection to the server queue manager, it is a **significant security exposure** and should be changed for systems which need to restrict which users may access the server queue manager.
3. The security exits may be used to pass the client userid to the server proxy. As mentioned earlier, the security exit data exchange is arbitrary and so may include a userid which can be used by the proxy. If the security exits are used, it is up to the SVRCONN exit to decide if the client userid should override any value in the MCAUSER parameter of the SVRCONN Channel definition.

It should be noted that the access control checks carried out for the proxy that use the ACLs associated with the server queue manager. If the client userid is passed to the server, measures should be taken to ensure that the capabilities of this userid are appropriately set up for that particular queue manager. This becomes especially important if the client is able to connect to several queue managers.

As explained above, the client server protocol is concerned with packaging and transporting MQ API requests and responses between client and server proxy. As such, messages are not explicitly passed across the connection. For this reason, there is no message exit available for the client server connection. Therefore, if any security functions are to be applied to the data passing between client and sever, this must be implemented in the send/receive exits. This may present a problem, as the data format passed between client and server is not exposed. This, in turn, makes it difficult to identify the data flows that contain *user* data – the part which most enterprises are interested in protecting. In order to ease this problem, the client/server data format is being partially exposed. The 10th byte of *all* client/server data exchanges identifies the

type of data segment being transmitted. From the viewpoint of the MQ API, the significant values for this byte are:

MQ Command	Value	Notes
MQCONN request	X'81'	1,2
MQCONN reply	X'91'	1,2
MQDISC request	X'82'	1
MQDISC reply	X'92'	1
MQOPEN request	X'83'	3
MQOPEN reply	X'93'	3
MQCLOSE request	X'84'	
MQCLOSE reply	X'94'	
MQGET request	X'85'	4
MQGET reply	X'95'	4
MQPUT request	X'86'	4
MQPUT reply	X'96'	4
MQPUT1 request	X'87'	4
MQPUT1 reply	X'97'	4
MQSET request	X'88'	
MQSET reply	X'98'	
MQINQ request	X'89'	
MQINQ reply	X'99'	
MQCMIT request	X'8A'	
MQCMIT reply	X'9A'	
MQBACK request	X'8B'	
MQBACK reply	X'9B'	

Notes:

These are not the only values of this byte. All other values are undocumented and **reserved**.

1. The connection between the client and server is initiated by the client application issuing MQCONN. Thus, for this command in particular there will be several other network flows.
   Similar considerations apply to MQDISC, which terminates the network connection.
2. MQCONNX is treated as MQCONN for the purposes of the client/server connection.
3. If a (quite large) distribution list is being opened, there may be more than one network flow per MQOPEN in order to pass all of the required data to the SVRCONN MCA.
4. If the message data exceeds the transmission segment size then there may be (possibly very) many network flows per single API command.

Message Level Security

Introduction

The MQSeries channels and the exits enable messages to be protected while in transit between queue manager components. However, messages will not be protected in any way while on queues – either in the source, intermediate or target systems. Because of this there is an increasing requirement, within messaging systems, to provide security functions at the message level, as opposed to providing functions at the queue manager or channel level. The functions that are required are as follows:

- Message level authentication
  Although the MQMD always contains a userid, there is no token present to authenticate that userid. Particularly if the message has come from a remote system, there may be a need to ensure that the userid in the MQMD is valid.
- Message integrity
  In order to guarantee that the message has not been altered since it was created by the putting application
- Message privacy
  In order to guarantee that the message cannot be viewed once it is created by the putting application
- Non-repudiation
  To provide both digital signature and digital receipt for the data in a message.

This section will look at some of the issues associated with message level security and some of the ways that these functions might be implemented in today's queue managers, which do not provide this level of function in the base products.

Issues Associated With Message Level Security

- The first (minor) issue with message level security is with naming. Message level security is synonymous with end-to-end security and application level security.
- The primary issue associated with message level security has been the fragmentation of the distributed security environment. Put simply, with the large number of different platforms available and different levels of security functions available, there has never been any assurance that a message encrypted on one platform could be decrypted on the receiving platform. In an homogeneous environment (for example, Windows NT), it is usually a valid assumption that similar facilities would be available on source and target. This is not true in a generalized, heterogeneous environment.
- If messages are protected in an end-to-end style, then there may be issues if messages are incorrectly routed and placed on a dead letter queue on some intermediate (or the sending) queue manager. Particularly, if the message is encrypted, only the target system is likely to be able to process (eg. re-route) the message data.

Support For Message Level Security

MQSeries does not currently support message level security functions. So, if an enterprise needs these functions, what is it to do? There are several options for existing enterprises, as follows:

- Use of MQSeries Channel exits
  While is it is (correctly) stated above that the channel exits do not provide true end-to-end protection of data, it would be possible to regard the exits as providing end-to-end function in the following manner; if an enterprise has determined that the communications infrastructure is the only exposure in the system, then use of the channel exits for end-to-end security could be justified.
  It is likely that enterprises like this are extremely rare!
- Each application could implement any message level functions that are required. This would be independent of the MQ API and the queue manager would treat the processed data just as it would treat unprocessed data. There are a few enterprises for whom this is an applicable (in fact the only possible) solution. These are usually enterprises that are *particularly* concerned with protection of their own data.
  It is clear that the above enterprises are in the minority!
  Any enterprise considering implementing protection of data in this way needs to be aware of data conversion considerations:
  - If security tokens (such as a remote authenticator or integrity token) have been added to the data then any data conversion to be performed must be aware of the presence of these tokens.
  - Integrity tokens are usually generated from a binary image of the data. Any data conversion of the data may invalidate the original integrity token.
  - If the data has been encrypted then any attempt to convert the data as a part of MQGET is unlikely to be successful.
- Many MQSeries enterprises are making use of high level APIs that encapsulate the MQ (and other) APIs. There are many reasons for this, though the most common are either to hide advanced MQ API functions from programmers or to add function to the MQ API. One of the functions that might be added to the MQ API is security. The relevant functions could be implemented ahead of MQPUT and after MQGET.
  This might be considered equivalent to implementing message level security within each application, as mentioned above. The primary difference is that a high level API is implemented and maintained centrally, rather than separately for each application. Note, however, that the considerations for application data conversion mentioned above still apply.
  There are vendor supplied products which provide high level APIs above the MQSeries API and provide this level of security function.

MQSeries Security Directions

Message Level Security

It is intended that MQSeries properly support message level security within the base product. This will require the ability to support the security functions mentioned in the previous section below the MQ API. There are two industry developments that make implementation of this function a viable proposition:

- As mentioned above, the distributed security environment has been fragmented, with no assurance of common function across a set of heterogeneous platforms. The Open Group has published a new security standard called **C**ommon **D**ata **S**ecurity **A**rchitecture, CDSA, which defines a common set of Public Key Infrastructure (PKI) facilities which are guaranteed to be available on any platform implementing CDSA. This standard is endorsed by Intel, Hewlett Packard, IBM and many others. IBM already has implementations of CDSA available on some platforms...the IBM Keyworks product.
  The Open Group specification for Common Data Security Architecture is ISBN 1-85912-194-2.
- One of the main difficulties with some existing security APIs (such as GSS-API) is that they are dependent upon a connection to the partner system. This means that they operate satisfactorily at the MQSeries channel layer (where they are used in the MQSeries V5 channel security exits) but are not useful for a general asynchronous message queuing model.
  The **I**nternet **E**ngineering **T**ask **F**orce, IETF, has been working on extensions to the GSS-API to provide support for asynchronous messaging applications. The IETF is defining **I**ndependent **D**ata **U**nit **P**rotection, IDUP, which encompasses all of the message level security functions detailed above. IDUP-GSS-API is documented by the IETF under the following document number draft-ietf-cat-idup-gss-07.txt and draft-ietf-cat-idup-cbind-03.txt .

While it would be possible to provide MQSeries message level security functions using only the APIs provided within CDSA, the task is made significantly simpler by the implementation of the IDUP-GSS-API on the CDSA APIs and this is the intention. It will provide the following facilities:

- Use of industry standard specifications to provide message level security functions
- Ability to ‘plug in’ various low level security functions (such as hardware or software encryption routines) below CDSA implementations.

Further, it is the intention to make the use of IDUP-GSS-API and CDSA implementations optional. This means that it will be possible to ‘plug in’ other message level security functions if CDSA is not the desired implementation.

MQSeries Access Control

There are a number of aspects related to access control that might be enhanced. Probably most important among these is the support of userids longer than 12 characters, most

common in the Windows NT environment. This issue is compounded in the NT environment by the possibility that the same userids might be present in multiple NT security domains with different characteristics.
The issue of supporting longer userids will be addressed within the queue manager. Windows NT V5 plan to address duplicate userids across multiple NT security domains by having unique userids in a managed NT environment.

Summary

MQSeries has a reputation within some (ill-informed) environments for having no security. While it is not suggested that MQSeries currently has a complete set of security functions, it should be clear from this document that MQSeries does implement a significant set of security functions – particularly with respect to access control – and does enable many other security functions to be added to the functions provided as a part of the base. It is a testament to the success of the MQSeries product family that independent software vendors are providing value-add security functions.
Further, there are MQSeries Development plans to address some of the deficiencies in MQSeries security functions, particularly in the area of message level security functions.