Mission:Messaging: Understanding WebSphere MQ authorization and the setmqaut command

T.Rob Wyatt

March 03, 2010

Profiles, entities, and groups, oh my
At its simplest, IBM WebSphere MQ authorization controls grant a set of privileges to a single entity for a single object. In practice however, the effective authorization granted to a particular user might draw from several access control entries (known as profiles in WebSphere MQ terminology) which might have been applied across multiple groups. The effective access is the combination of all applicable profiles, from all applicable groups or accounts. To understand all of this, you have to look under the covers at the interactions between profiles, accounts, and groups, as well as the tools used to manage these settings.

To illustrate the different interactions between profiles and entities, let's look at three use cases:

• In the first use case, an application will connect, issue a request, and wait for a reply. This simple model can be built from a few profiles.

• The second use case is that of a connection from an adjacent queue manager. Here, you require access to put messages onto all user-defined queues but without also granting the adjacent queue manager administrative rights. This case will illustrate how different access control lists interact with a given set of queues.

• The final use case illustrates how access is accumulated from several sources. For this scenario, a hypothetical developer will transition off of the "Foo" project and onto the "Bar" project. Since the Foo and Bar applications interact with one another, they both require access to some of the same queues, but with different rights. You will see how the access rights change over time as the group memberships associated with the developer's account are updated.

Before proceeding further, it is necessary to define the term entities in the context of this discussion. Authorizations can be applied at several levels. The lowest of these is an individual account. This account might belong to a human user, an application, system tools, instrumentation agents, or monitors. Typically, an account owned by a human user is called a user.
ID and an account used by applications or instrumentation is known as a service account. When the distinction does not matter, the account is referred to more generically as a principal, hence the –p option in the setmqaut command.

Authorizations can also apply at the group level. A group represents a collection of zero or more principals. In some models, such as Windows® Active Directory, it is even possible to grant authorizations to local groups containing domain groups, which contain principals.

In this discussion, an entity is anything to which an authorization rule can be applied. Depending on the platform and the account management tools, the entity that you grant authorizations to could be an individual account or a local group. Where the distinction between principals and groups doesn't matter, the object in question is referred to as an entity.

To summarize:

- User IDs and service accounts are types of principal.
- Principals and groups are types of entity.

For the remainder of the article, I will use the most specific applicable term.

Use case #1: Authorizing an application

Recall that in this case, an application requires the ability to connect to the queue manager, put a message to a queue, and get a reply. The authorizations that are granted are the same for a remote application connecting over a SVRCONN channel as they are for a local application connecting in bindings mode. The only difference is where WebSphere MQ obtains the principal.

For a local bindings mode connection, the principal is the account associated with the process that is connecting to the queue manager. Because the connected process is running locally, WebSphere MQ can obtain the process owner by looking in the process table. The account can be trusted because it has been authenticated by the operating system.

For remote applications using a client channel connection, the principal is the account in the channel's MCAUSER attribute. This account can be trusted if the MCAUSER is hardcoded into the channel definition or authenticated and set by a security exit.

Given an application service account apple that is a member of the fruit group, the commands to grant the required authorizations might look something like this:

```
setmqaut -m VENUS -t qmgr -g fruit -all +inq +connect
setmqaut -m VENUS -t queue -g fruit -n FRUIT.REQUEST -all +put +inq
setmqaut -m VENUS -t queue -g fruit -n FRUIT.REPLY -all +get +browse +inq
```

Specifying the queue manager

The –m option is also used to specify the queue manager to which the command will apply. If there is a default queue manager defined it would be possible to omit the –m option, but it is not advisable to do so. Where security is concerned, it is best to use the most explicit command syntax that achieves the desired result. Specifying the –m option requires knowledge of the
intended target queue manager. Omitting the option requires the same knowledge, but also requires the administrator to know precisely which local queue manager is currently set as the default. It is simpler -- and also more reliable -- to explicitly specify the queue manager using the –m option in all cases.

Authorizing groups and principals

The next parameter in the example is –g which is used to specify the group. Although the setmqaut command accepts a –p option to specify an account, the –g group option is usually the one you want. The reason for this is that the –p option behaves differently depending on the platform. On UNIX® and Linux® platforms, the queue manager will look up the principal specified in the –p option, obtain the primary group for the specified principal and then apply the profile to that group. If the principal has staff as its primary group and apple as the secondary group, then staff is the group that will be authorized when the –p option is used.

I have seen a number of cases in which the –p option was used on UNIX platforms and the primary group of the account changed over time. Each time the authorizations were updated after the group membership change, the profile was applied to the then current group. When this happens, the application does not fail so there is nothing to alert an administrator that excessive authorizations have been granted. Specifying the –p option on platforms other than Windows executes an implicit action rather than an explicit action. As noted earlier in the discussion about default queue managers, explicit actions are always preferred.

On Windows platforms, the –p option behaves differently. Here, it can actually resolve to an individual account. However, it might not always resolve to the account that was intended so it is still necessary to take care when using the –p option on Windows hosts.

As an example, consider how Windows will resolve an inquiry on a principal named applid. First, the local Security Account Manager (SAM) database is searched for a matching account. If a local account is found, that is always used. But if no local account is found, the Windows server will inquire in all of the domains that it trusts. The order of this domain search is undetermined, so if a matching account exists in more than one domain, it is not possible to know in advance which instance of the account will resolve the inquiry.

This is significant because when the setmqaut command is executed, the profile created contains the Security ID (SID) rather than the string version of the account name. The SID is a globally unique identifier for a specific account in a specific domain. If applid exists in DomainA and also in DomainB, then each instance will have a unique SID associated with it. The setmqaut command will receive the SID of whichever instance happened to resolve when the command was executed. This may or may not be the same instance that is returned when the account is looked up at run time.

The ambiguity can be resolved when specifying a principal by qualifying it with the desired domain name. The correct format for setmqaut commands and the MCAUSER channel attribute is user@domain, where domain is either an actual domain name or the local host name. When this
format is used, Windows will correctly resolve the principal in the desired domain, even when an account with the same name resolves in multiple locations.

However, groups cannot be specified in this manner. The only groups that WebSphere MQ recognizes are those defined locally on the server where the queue manager is hosted. Therefore, if authorization on Windows queue managers is to be based on groups, the principal in question must either belong to the local group, or belong to a domain group which is a member of the local group.

For these reasons, I recommend using the `setmqaut -p` option only on Windows hosts, and then fully qualify the account using the format of `user@domain`. Here is an example that illustrates the recommended use of the `–p` option:

```bash
setmqaut -m VENUS -t qmgr -p user1@mqhost -all +inq +connect
```

**Specification of rights**

The last parameters in the `setmqaut` command are the permissions that will be granted. Notice that the examples provided here all contain `-all`, followed by additional rights such as `+inq` and `+connect`. The reason for this is that `setmqaut` commands are cumulative.

For example, consider the results of these commands:

```bash
C:\>setmqaut -m VENUS -t qmgr -g fruit +dsp
The setmqaut command completed successfully.

C:\>setmqaut -m VENUS -t qmgr -g fruit +inq +connect
The setmqaut command completed successfully.

C:\>dspmqaut -m VENUS -t qmgr -g fruit
Entity fruit has the following authorizations for object VENUS:
  inq
  connect
  dsp
```

Notice that the results of the first two commands were combined. When `–all` is added to the second command, it replaces the authorizations previously set rather than adding to them.

```bash
C:\>setmqaut -m VENUS -t qmgr -g fruit +dsp
The setmqaut command completed successfully.

C:\>setmqaut -m VENUS -t qmgr -g fruit +inq +connect
The setmqaut command completed successfully.

C:\>dspmqaut -m VENUS -t qmgr -g fruit
Entity fruit has the following authorizations for object VENUS:
  inq
  connect
```

Usually what is intended is that the `setmqaut` command replaces any previous profile rather than add to it. The `–all` parameter ensures that this is in fact what happens.
The importance of +inq

Notice that +inq was provided for all profiles in the examples above. This is because some of the attributes of WebSphere MQ objects exist specifically to pass information to applications using these objects. For example, the BOQNAME attribute can hold the name of a backout queue into which problem messages can be placed. The BOQTHRESH attribute contains the number of times a message can be backed out before being placed into the backout queue.

The inquiry occurs in many cases, even if it is not under control of the application code. For example, the JMS classes will inquire on the BOQNAME and BOQTHRESH whether or not the application does this explicitly. As a general rule, anything that is legitimately permitted to connect to the queue manager should be granted +inq rights on the queue manager and any objects it opens.

Summing up use case #1

In this first use case, a single group fruit was granted access to connect to the queue manager, put and inquire on the request queue, and get, browse, and inquire on the reply queue. In this example, there is a one-to-one correspondence between a profile and the object to which it applies.

The default for any principal that is not in the mqm group is to have no access. After running the three commands, members of the fruit group are restricted to the specified actions. No other activity is possible for principals whose only membership is to the fruit group.

Use case #2: The adjacent queue manager

This use case is one of the cornerstones of basic WebSphere MQ administrative hardening. The objective is to grant the adjacent queue manager very broad rights to place messages onto user-defined queues on the local queue manager, but without also granting administrative access.

Channels from other queue managers, known as MCA channels in WebSphere MQ parlance, come in three varieties: RCVR, RQSTR, and CLUSRCVR. As you might know, all inbound channels run with full authority of the queue manager and thus have access to all queues. In order to restrict access, it is necessary to force the channel to use a low-privileged account for authorizations. This is accomplished by setting the MCAUSER value of the channel to the appropriate account.

Any account that is dedicated for this purpose will do, but I like to pick an account name that is easily recognized as administrative along with being self-documenting. For these reasons, I've been using the account mqmmca for RCVR, RQSTR, and CLUSRCVR channels. The mqm part draws an association with the mqm account that is used to administer WebSphere MQ. Hopefully, account administrators already treat the mqm account and group as administrative and restrict access to these. Additional accounts and groups named mqm* should intuitively also fall under the same controls. The mca part of the account name indicates that the account is specific to MCA channels. I use a similar account called mqmmqi for MQI channels, more commonly known as client or SVRCONN channels. The two types of channel have slightly different access
requirements so it is important to create different accounts and groups to run them under. This is explained further in the next use case.

So, given an account and group named mqmmca, how do you go about authorizing it to a large number of queues? The account has no access unless profiles are created for it, but using the techniques from the first case study would require a profile for every queue to which the channel needs access. That is not only labor-intensive, it is also fragile because every new queue requires a new profile; forget to create the profile and the application breaks. A better approach would be to establish a default policy of allow-all so that the only profiles required are for the few queues that are considered administrative. Fortunately, setmqaut has a provision for this. The use of wild-card characters in the profile name creates what are known as generic profiles.

**Generic profiles**

WebSphere MQ object names are customarily composed of nodes delimited by dot characters. The setmqaut command enables the use of a question mark to replace a single character in a name or an asterisk to replace zero or more characters in a node. To use the example from the manual: AB.?D matches objects named AB_CD, AB_DD, AB_ED and so on.

The asterisk matches zero or more characters delimited by a dot character. For example, ABC.*.JKL applies to the objects ABC.DEF.JKL, and ABC.GHI.JKL. Notice that the profile matches a queue named ABC..JKL because it has three nodes, even though the middle node has no characters. But ABC.*.JKL does NOT match a queue named ABC.JKL which has only two nodes in the name.

The question mark and asterisk can be used more than once in the same profile. For example, the profile ???.*.* matches objects with three characters in the first node of the name and three nodes total in the name.

A wild-card consisting of two asterisks matches zero or more whole nodes and can be used at the beginning, middle, or end of a name. Valid examples include:

**Table 1. Wild card characters in setmqaut commands**

<table>
<thead>
<tr>
<th>**.REPLY</th>
<th>Matches all names with a final node of .REPLY.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAY.*</td>
<td>Matches all names beginning with the node PAY.</td>
</tr>
<tr>
<td>PAY.*.REPLY</td>
<td>Matches all names with first node of PAY, a final node named .REPLY, and any number of intermediate nodes.</td>
</tr>
<tr>
<td>**</td>
<td>Matches all names.</td>
</tr>
</tbody>
</table>

Some examples will help illustrate this better. Here, the double-asterisk is used to represent zero or more nodes at the beginning of a queue name. Be aware that the double asterisk matches zero or more nodes, not one or more nodes. This accounts for the final example showing that the queue REPLY, with but one node in the name, is authorized.
In the next example, the double asterisk substitutes for multiple nodes in the middle of a name.

C:\>setmqaut -m VENUS -t queue -n **.REPLY -g fruit -all +inq +dsp
The setmqaut command completed successfully.

C:\>dspmqaut -m VENUS -n .REPLY -t queue -g fruit
Entity fruit has the following authorizations for object .REPLY:
  inq
dsp

C:\>dspmqaut -m VENUS -n ..REPLY -t queue -g fruit
Entity fruit has the following authorizations for object ..REPLY:
  inq
dsp

C:\>dspmqaut -m VENUS -n REPLY -t queue -g fruit
Entity fruit has the following authorizations for object REPLY:
  inq
dsp

Notice that in the last example, PAY.**.REPLY did not match the name PAYREPLY because the profile requires at least two nodes named PAY and REPLY, separated by at least one dot character. Similarly, the profile PAY**REPLY is not valid because the double asterisk must be delimited when used in combination with other characters.

**Overlapping profiles**

The ability to use generic profiles opens the possibility that multiple profiles will apply to the same object. For example, the profiles PAY.** and PAY.**.REPLY and PAY.ROLL.REPLY can all apply to a queue named PAY.ROLL.REPLY. The rule in this situation is that the most specific profile that applies to the named object is the one used for authorization checks. In this case, the profile with no wild cards is the most specific and would be the one to take precedence. Of the two remaining
profiles, PAY.**.REPLY takes precedence over PAY.** because any matching name will have matched on five specific characters as opposed to three.

It is this ability to use overlapping profiles that enables you to build an authorization scheme for the second use case. The channel requires one profile to connect to the queue manager, another to establish access to all queues, and then several specific profiles that further restrict access to queues that are considered administrative. That set of profiles would look like this:

```
setmqaut -m VENUS -g mqmmca -t qmgr -all +connect +inq +setall
setmqaut -m VENUS -g mqmmca -n '**' -t queue -all +put +setall
setmqaut -m VENUS -g mqmmca -n SYSTEM.ADMIN.COMMAND.QUEUE -t queue -all +none
setmqaut -m VENUS -g mqmmca -n SYSTEM.DEFAULT.INITIATION.QUEUE -t queue -all +none
setmqaut -m VENUS -g mqmmca -n SYSTEM.CLUSTER.TRANSMIT.QUEUE -t queue -all +none
```

The '**' profile establishes rights for the channel to put messages onto any queue. Because the remaining profiles are more specific, they override this profile so that the channel cannot put messages onto the command queue, the default initiation queue, or the cluster transmit queue.

This is the bare minimum that will restrict administrative access from an adjacent queue manager and is only used here for illustration. In practice, this authorization scheme would also restrict access to any user-defined initiation queues, transmit queues, and many of the SYSTEM.** queues. If the queue manager participates in a cluster, access to SYSTEM.CLUSTER.COMMAND.QUEUE by CLUSRCVR channels is required. On the other hand, access to SYSTEM.ADMIN.COMMAND.QUEUE is almost never required by an adjacent queue manager.

**Use of the +setall privilege**

MCA channels require +setall rights on the queue manager and on queues. The reason for this is that the channel agent will preserve the contents of the message descriptor by setting all the fields to match the message from the sending queue manager. These include the message PUT date and time, the application name, and the user ID. This differs from ordinary applications that are not trusted to set the message context fields and instead inherit these from whatever values the queue manager deems appropriate. So +setall is required for any channel of type RQSTR, RCVR or CLUSRCVR.

Although the MCA channel agent is as trusted as the other internal queue manager components, the MQI channel agent is not. This is because the MQI channel agent is driven by the remote application which has access to the entire WebSphere MQ API. Although there are cases where a remote application legitimately needs to set the message header fields, this is the exception rather than the rule. As a rule, +setall rights should not be granted to channels of type SVRCONN.

**Summing up use case #2**

The adjacent queue manager was authorized using a single account in a single group. Had this example been on a Windows host, it would have been possible to omit the group and authorize the principal without making it a member of a group. In either case, the technique used is to overlap profiles to establish first a default policy of allow-all, and then to override specific queues with tighter restrictions. It is this ability to use generic profiles that overlap which enables automatic
enforcement of policies against newly created queues with very fine grained ability to specify exceptions.

**Use case #3: The roaming developer**

In the previous use case examples, the rights granted to a principal were derived from one and only one profile. When two profiles matched the same object, the most specific profile took precedence and completely overrode the rights specified in the less specific profile. This next example illustrates that rights can accumulate from multiple profiles when the principal involved is a member of multiple groups and these groups have differing access rights.

For this example, a hypothetical developer will have a user ID **alice**. Since Alice is working on the Foo project, the alice User ID is currently in the "foo" group. The Foo application will request services from the Bar application and expect replies. Therefore, Alice should have get access to queues named FOO.** and limited access to queues named BAR.**.RQST. This access is granted using these profiles:

```
setmqaut -m VENUS -t qmgr -g foo -all +inq +connect
setmqaut -m VENUS -t queue -g foo -n 'FOO.*' -all +inq +get +browse
setmqaut -m VENUS -t queue -g foo -n 'BAR.**.RQST' -all +put
```

Since the Foo application is the primary user of the Bar service, and since Alice is the resident expert on all things Foo, the project manager of the Bar application hires her as the new Quality Assurance lead. As part of the transition, the alice user ID is added to the bar group which has these permissions:

```
setmqaut -m VENUS -t qmgr -g bar -all +inq +connect
setmqaut -m VENUS -t queue -g bar -n 'BAR.**.RQST' -all +get
setmqaut -m VENUS -t queue -g bar -n 'FOO.**.REPLY' -all +put
```

At this point, Alice's user ID has been granted both +get and +put on queues named BAR.**.RQST by virtue of belonging to two different groups. The effect is cumulative as seen with the dspmqaut command:

```
setmqaut -m VENUS -t queue -g foo -n 'BAR.**.RQST' -all +put
setmqaut -m VENUS -t queue -g bar -n 'BAR.**.RQST' -all +get
dspmqaut -m VENUS -t queue -n BAR.UPDATE.RQST -p alice
Entity alice has the following authorizations for object BAR.UPDATE.RQST:
  get
  put
```

In the example above, the profiles specified in the two setmqaut commands were the same: BAR.**.RQST. But the profiles to authorize the FOO queues were of different specificity:

```
setmqaut -m VENUS -t queue -g foo -n 'FOO.*' -all +inq +get +browse
setmqaut -m VENUS -t queue -g bar -n 'FOO.**.REPLY' -all +put
```
The FOO.**.REPLY profile is more specific than the FOO.** profile. But the rule of the most specific profile does not apply in this case. Because the two profiles are in two different groups, they do not actually overlap. This is evident from the results of the dspmqaut command:

```bash
dspmqaut -m VENUS -t queue -n FOO.BAR.REPLY -g foo
Entity foo has the following authorizations for object FOO.BAR.REPLY:
  get
  browse
  inq

dspmqaut -m VENUS -t queue -n FOO.BAR.REPLY -g bar
Entity bar has the following authorizations for object FOO.BAR.REPLY:
  put

dspmqaut -m VENUS -t queue -n FOO.BAR.REPLY -p alice
Entity alice has the following authorizations for object FOO.BAR.REPLY:
  get
  browse
  put
  inq
```

**The dspmqaut command**

The **dspmqaut** command used in the previous examples calculates the effective permissions based on all applicable profiles. Although this is extremely useful, there are occasions when authorizations combine from several profiles to produce unexpected results and it is not obvious why an authorization is failing. Fortunately there is a command for this situation as well:

```bash
dmpmqaut -m VENUS -t queue -n FOO.BAR.REPLY  -p alice -e
profile:     FOO.**.REPLY
object type: queue
entity:      bar
entity type: group
authority:   put
---
profile:     FOO.**
object type: queue
entity:      foo
entity type: group
authority:   get browse inq
```

The **dmpmqaut** command with the **-e** option displays all of the profiles that contribute to an entity’s effective cumulative rights to an object. Here, you see that Alice’s user ID has inherited the rights granted to both the foo and the bar groups and that the most specific profile from each group has been applied to calculate effective rights on the FOO.BAR.REPLY queue.

**Debugging authorization problems**

Despite your best efforts, there are times when your applications will receive **2035 Authorization Error** return codes. Usually, this is because an object has not been authorized, which is easy to spot. The more difficult case is when the application has the correct object but it attempts to use an elevated right that has not been granted, such as +altuser or +passid. When this occurs, the easy way to diagnose the problem is to enable authorization events.

If you have been working with WebSphere MQ for many years, the mention of authorization events might make you groan and reach for your hexadecimal calculator. Those days are long gone! The MS0P SupportPac includes a PCF decoder plug-in that makes short work of parsing
event messages. The event message will tell you the user ID that made the failing call, the object that failed, the API call made against that object, and the options used in the API call. If it is not immediately clear what went wrong, you can use this information along with the dmpmqaut command to display a list of all the profiles that contributed to the effective rights for that user and that object.

SupportPac MS0P also now contains a chainable Object Authority Manager (OAM) implementation that logs calls and then passes them to the native OAM. The OAM is designed as a pluggable component that can be replaced or augmented in order to extend or replace the native functionality. The implementation in MS0P can be used as a skeleton for your own custom implementation, should you wish to write a new one. If this sounds familiar, you might recognize it as having previously been in SupportPac MS07.

**Putting it all together**

Obviously, the ability to specify different levels of wild cards and accumulate rights from multiple groups provides the means with which to build extremely fine-grained authorization schemes. My advice is to avoid these if at all possible. Complexity and security are mortal enemies. Always choose the simplest solution that solves the problem. In fact, I would go so far as to say that the authorization scheme is not complete until there is nothing that you can remove or simplify without breaking it.

I should also mention that the objective of this article was to show how the pieces interact rather than to provide workable authorization examples. That is why topic authorizations were not used in the illustrations. Although they use the same syntax as queue authorizations and the same precedence rules for profile specificity apply, authorizations on topics operate on a namespace rather than on specific objects. These will be the subject of a future installment of *Mission: Messaging*.

My other piece of advice is that authorizations are not effective unless authentication has been performed. Recall that authorization enforces what you can do whereas authentication first establishes who you are. When connections are made to the queue manager, either from other queue managers or on client channels, it is up to the administrator to determine the level of authentication required and configure that.

Keeping that in mind, any inbound channel that does not have an MCAUSER set enables administrative access to the queue manager. The best authorization scheme in the world is completely useless if non-administrative entities can connect with full administrative rights or spoof the identity of legitimate applications or users. Make sure that your authentication is in place for all of your inbound channels before spending a lot of time on authorization schemes. This includes channels named SYSTEM.DEF.** and SYSTEM.AUTO** as well. Enabling SSL, a security exit, or some combination of the two on every inbound channel is usually required to assure that your authorization scheme is enforced properly.
WebSphere MQ security: Live!

I will be presenting two WebSphere MQ security sessions this year at IMPACT 2010. The first is a base administrative hardening session, which covers authentication in a lot more detail than we did in this article. The other session is the brand new WebSphere MQ security lab which will provide hands-on experience configuring SSL and security exits to provide authentication, as well as building authorization schemes and diagnosing problems. If you will be at IMPACT this year, please attend one of the lab sessions and let me know what you think. I look forward to meeting you in Las Vegas!
Related topics

- WebSphere Flashes
- MS0P: WebSphere MQ Explorer - Configuration and Display Extension Plug-ins
- WebSphere MQ SupportPacs main page
- Commands
  - dmpmqaut
  - dspmqaut
  - setmqaut
- Podcast: The Deep Queue
- Author's Web page: T-Rob.net

© Copyright IBM Corporation 2010
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)