FROM THE ARCHIVES: Getting going without turning off IBM MQ Security

Naomi_S
Published on 01/10/2018

We have a wealth of useful blogs in our archive on MQDev. This one disappeared mysteriously, so we decided to re-publish it here! Enjoy.

Written by Morag Hughson.

Getting going without turning off IBM MQ Security

You have noticed that the IBM MQ product takes a stance to have security features turned on by default. many of you will have undoubtedly tripped over CHLAUTH rules blocking you because you asserted a privileged user ID over a client connection, or when playing with V8 or a more recent version, you may have also tripped over CONNAUTH rules mandating you provided a user ID and password because you asserted a privileged user ID over a client connection.

You'll have no doubt learned that to just turn off CHLAUTH and CONNAUTH completely, you can issue the following commands:

```
ALTER QMGR CHLAUTH(DISABLED) CONNAUTH(' ')

REFRESH SECURITY TYPE(CONNAUTH)
```

Plus, adding the user ID that is already defined on the queue manager system and in the privileged mqm group to your server-connection channel's MCAUSER field.

```
ALTER CHANNEL('SYSTEM.DEF.SVRCONN) CHLTYPE(SVRCONN) MCAUSER('morag1')
```

However, as you do so, you wish that you didn't have to do this and that there was a simple little script you could run to enable your own user to get in without needing to open up the queue manager completely by turning them off. That's what this post hopes to teach you. This could be useful to demonstrate MQ or to play with it and get to know it, but all the while maintaining the security of your system, or even getting to the point where you can demonstrate the security even though you're still just learning how to use it.

Changes to make to keep CHLAUTH and CONNAUTH enabled

There are several steps to this which I'm going to go through one at a time explaining why they are there, and then at the end I'll supply all the commands together (so you can copy

them into a script if you want). I'm going to describe two patterns here, one where you want to supply a user ID and password and use a privileged user ID, and the other one where you don't want to supply a user ID and password and will use a non-privileged user ID.

Pattern 1: Password authenticated privileged user ID

In this pattern we will use your existing privileged user ID for the client connections you make into the queue manager, so you need to know what that user ID is and the password for that user ID. In my example, the privileged user ID is 'morag1' and my queue manager is called MORAGQM running on IP address 9.10.11.12 with port number 1515. You'll have to replace those details below with your own user ID and queue manager details. This pattern assumes your queue manager is running with V8 of IBM MQ.

We've chosen to make use of the V8 feature Connection Authentication and supply a user ID and password when we connect to the queue manager. By default CONNAUTH is enabled and is set to REQDADM for client connections which means privileged users are REQUIRED to provide a password but non-privileged users don't have to. So we don't need to change any settings there, as we're going to follow that pattern.

CHLAUTH will catch us out though as we're supplying a privileged user ID. So, as described in CHLAUTH – Allow some privileged admins, we're going to create one route into the queue manager where the privileged user ID is allowed in since it is validated by password. To follow good practice, we're going to do this by creating a new server-connection channel rather than using one the pre-existing SYSTEM ones, but you might equally do this with the SYSTEM.ADMIN.SVRCONN that MQ Explorer uses by default. I've called this server-connection channel "PASSWORD.SVRCONN" to go with this blog post, but you might equally name it related to your department or application name.

```
DEFINE CHANNEL(PASSWORD.SVRCONN) CHLTYPE(SVRCONN)
```

Having defined it, I am now going to set up a CHLAUTH rule to allow a privileged user to be able to come through on this channel. This rule over-rides the default CHLAUTH rules that blocks all *MQADMIN – i.e. privileged users. It is over-ridden only for my specific channel, so the use of a privileged user ID is blocked over any other channel.

```
SET CHLAUTH(PASSWORD.SVRCONN) TYPE(BLOCKUSER) USERLIST('nobody')
DESCR('Allow privileged users on this channel')
```

We also follow the best practice of setting up a backstop rule (as described in CHLAUTH – the back-stop rule) so that no connections can be made through this, or any other, channel; and then we allow connections into our channel, but only when a password is supplied.

```
SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
DESCR('BackStop rule')

SET CHLAUTH(PASSWORD.SVRCONN) TYPE(ADDRESSMAP) ADDRESS('*')
USERSRC(CHANNEL) CHCKCLNT(REQUIRED)
```

For the final step, I need to make sure that the client applications I run use the user ID I supplied (which has been password validated). This is not set by default to aid migrations from systems that are relying on the runas user ID which is flowed from the client machine

for all their profile checking. If you run your client application under the same user ID as you supply in the user ID and password on your application, these last two commands are redundant.

```
ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS)
ADOPTCTX(YES)

REFRESH SECURITY TYPE(CONNAUTH)
```

To test this out, you can use the runmqsc program in client mode and supply the user ID and password with it. More details in Bitesize Blogging: MQ V8 – Client MQSC. Type the following on your client machine:

```
set MQSERVER=PASSWORD.SVRCONN/TCP/9.10.11.12(1515)

runmqsc -c -u morag1 MORAGQM
```

Before allowing you to enter any commands, runmqsc will prompt you for the password.

You can make use of various MQ samples which have been updated for user ID and password supplying, in order to try out putting and getting messages. You can read more about that in Bitesize Blogging: MQ V8 – Samples can use user ID and password.

Here's the full script for Pattern 1:

```
DEFINE CHANNEL(PASSWORD.SVRCONN) CHLTYPE(SVRCONN)

SET CHLAUTH(PASSWORD.SVRCONN) TYPE(BLOCKUSER) USERLIST('nobody')
DESCR('Allow privileged users on this channel')

SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
DESCR('BackStop rule')

SET CHLAUTH(PASSWORD.SVRCONN) TYPE(ADDRESSMAP) ADDRESS('*')
USERSRC(CHANNEL) CHCKCLNT(REQUIRED)

ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS)
ADOPTCTX(YES)

REFRESH SECURITY TYPE(CONNAUTH)
```

Pattern 2: IP address filtered non privileged user ID

The first thing we notice with the default values of CHLAUTH and CONNAUTH is that they are both determined to keep out privileged user IDs from client connections. As you saw in pattern 1, you can make some changes to allow in a privileged user ID and have that protected by user ID and password validation. Alternatively, you can work with non-privileged user IDs and learn to grant only those accesses the user IDs need. Doing this can become as natural as defining the queue, and can easily be done from the same script.

The first step here is to create a user ID and ensure it not a member of the mqm group. In my example, I created a group called 'nonprivmq' and had these users be a member of that

group. The commands to do that differ from platform to platform and you probably know them as well as I do, but here are a few examples:

Windows:

```
net localgroup nonprivmq /ADD
net user morag2 passw0rd /ADD
net localgroup nonprivmq morag2 /ADD
```

Linux:

```
groupadd nonprivmq
useradd morag2 -g nonprivmq
```

This user ID that you have created on the queue manager system is not privileged, so it has access to nothing. A connection asserting that user ID would be allowed in by CHLAUTH rules but then would fail because it didn't even have +connect authority to the queue manager. Rather than having to make changes at the client side, or have to define the user ID on the client machine, we are just going to map our connections to this user ID when the server-connection channel runs. This mapping will be done just by filtering by IP address of the client machine – not really authentication, but better than nothing, and will give you a feel for how to use CHLAUTH rules. For decent security you should consider user IDs and passwords (which were shown in Pattern 1) or the use of Digital Certificates and TLS on your channel connections.

As above, we are going to create a new, non SYSTEM server-connection channel for our demonstration. This time I will name it NONPRIV.SVRCONN, although again you will probably want to name it for your department or application.

```
DEFINE CHANNEL(NONPRIV.SVRCONN) CHLTYPE(SVRCONN)
```

Having defined it, we will create our CHLAUTH rules, a backstop rule just as in Pattern 1, and a single IP address rule that allows only your client machine to connect and maps the connection to use your newly created non-privileged user ID.

```
SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
DESCR('BackStop rule')

SET CHLAUTH(NONPRIV.SVRCONN) TYPE(ADDRESSMAP) ADDRESS('9.10.20.30')
MCAUSER('morag2')
```

We're not finished yet, but if you were to connect into that server-connection channel now:

```
set MQSERVER=NONPRIV.SVRCONN/TCP/9.10.11.12(1515)
```

you'd find that you still didn't have access, not because CHLAUTH blocked you, but because you don't have the necessary privileges required to do anything, like connect for example.

The next step is to rectify that. Since the premise here is that this one user is to be able to do everything, the following set of authority commands are very broad. While this is fine for

trying out MQ, in a production system you will of course only grant users access to the resources they need, and not to everything as the following commands do.

```
SET AUTHREC OBJTYPE(QMGR) GROUP('nonprivmq') AUTHADD(ALL)

SET AUTHREC OBJTYPE(QUEUE) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(QUEUE) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(CHANNEL) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(CHANNEL) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)
```

These pairs of commands can be repeated for each object type to ensure access to absolutely everything is granted to the ‘nonprivmq’ group. Alternatively, if you want to start using security in a more granular fashion, as you will do in production, you can change the PROFILE string to only contain the name of the object you want access to, or a prefix that covers all your application’s objects, e.g. INVOICE.**. If your queue manager is only being used for client applications and not client administration, you could omit all the AUTHADD(CRT) rules since your applications aren’t going to need those.

Here’s the full script for Pattern 2:

```
DEFINE CHANNEL(NONPRIV.SVRCONN) CHLTYPE(SVRCONN)

SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS)
DESCR('BackStop rule')

SET CHLAUTH(NONPRIV.SVRCONN) TYPE(ADDRESSMAP) ADDRESS('9.10.20.30')
MCAUSER('morag2')

SET AUTHREC OBJTYPE(QMGR) GROUP('nonprivmq') AUTHADD(ALL)

SET AUTHREC OBJTYPE(QUEUE) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(QUEUE) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(CHANNEL) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(CHANNEL) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(CLNTCONN) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(CLNTCONN) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(AUTHINFO) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(AUTHINFO) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(COMMINFO) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(COMMINFO) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(LISTENER) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(LISTENER) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(NAMELIST) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(NAMELIST) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(PROCESS) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(PROCESS) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)
```

```
SET AUTHREC OBJTYPE(SERVICE) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(SERVICE) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(TOPIC) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
SET AUTHREC OBJTYPE(TOPIC) PROFILE('**') GROUP('nonprivmq') AUTHADD(CRT)

SET AUTHREC OBJTYPE(RQMNAME) PROFILE('**') GROUP('nonprivmq') AUTHADD(ALL)
```

Summary

Hopefully, this post will encourage you to use your queue managers without turning off security completely, thus preparing you and teaching you how MQ security works before you get to the point of running a production system. Clearly the above patterns are still fairly open in that the user we have allowed in, in both cases, can do anything. However, it is a good starting pattern for your test queue managers, or sandbox queue managers where you want to get a feel for how things work. Then you can try it out with more granular access for your business applications as you develop them.

For more information on MQ Security, please read The interaction of CHLAUTH and CONNAUTH in IBM MQ.