MQ update

71

May 2005

In this issue

A tc PUBLICATION

MQ Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs $380.00 in the USA and Canada; £255.00 in the UK; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 2000 issue, are available separately to subscribers for $33.75 (£22.50) each including postage.

Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

MQ Update on-line

Code from *MQ Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at www.xephon.com/mq; you will need to supply a word from the printed issue.

WebSphere MQ Extended Security Edition

IBM WebSphere MQ Extended Security Edition is a single package that consolidates WebSphere MQ Version 5.3 and IBM Tivoli Access Manager for Business Integration (TAMBI). Basically this provides you with an option to implement application-level data protection without changing or even re-compiling your existing WebSphere MQ applications.

WebSphere MQ Extended Security Edition enables a high level of security for sensitive transactions across the messaging environment, irrespective of the platforms that the message traverses. It provides an end-to-end, application-level, data protection model and allows the administrators to perform enterprise-wide remote management of security policies on queues and queue managers.

WHAT IS TAMBI?

TAMBI is an integrated component of the IBM identity management solution that provides:

1 Identity life-cycle management (user self-care, enrolment, and provisioning).

2 Identity control (access and privacy control, single sign-on, and auditing).

3 Identity federation (sharing user authentication and attribute information between trusted Web services applications).

4 Identity foundation (directory and workflow).

TAMBI also provides an authorization plug-in for the IBM WebSphere Business Integration brokers, which replaces the native authorization services provided earlier. TI shares the following set of services with other Tivoli Access Manager products including TAM for eBusiness and TAM for Operating Systems:

1 Central Security Policy Manager
2 Central Credential directory.
3 Web-based administration tool.

This new plug-in and the common set of services together simplify the administration process. Now, the security policy across WebSphere MQ queues, WebSphere BI broker publish/subscribe topics, Web resources, and Unix/Linux resources can be consolidated and managed via a single administration tool.

To use TAMBI, it is not necessary to license the Tivoli Enterprise Management Framework.

WEBSPHERE MQ AND TAMBI

The link and channel-level data protection offered by WebSphere MQ ensures the integrity and confidentiality of the messages only when the message is in transit. For applications that use WebSphere MQ for highly-sensitive data (like personal information and high-value financial transactions), an end-to-end security solution is required. With additional application-level data protection, this could be achieved without complex security-related coding.

TAMBI provides application-level data protection for WebSphere MQ-based applications, so that sensitive messages can be digitally signed and encrypted even before they get to WebSphere MQ, so that the integrity and confidentiality of the messages when under the control of WebSphere MQ can be ascertained.

Customers currently having WebSphere MQ can upgrade to this Security edition of WebSphere MQ by getting a licence for TAMBI. TAMBI supports systems running IBM WebSphere MQ Server as well as systems running WebSphere MQ Client. Applications written using the WebSphere MQ Native API as well as the JMS API are supported.

TAMBI Host Edition provides extended security services for mainframe applications using WebSphere MQ for z/OS. Also for the mainframe SSL, IBM Policy Director Authorization Services for
z/OS and RACF or equivalent should be installed. TAMBI Host edition supports WebSphere MQ applications running under CICS and IMS/DC, as well as batch.

HOW DOES APPLICATION DATA PROTECTION WORK?

Queue put/get permissions

TAMBI allows you to define policies that control which applications can **put** or **get** messages from specific queues or queue managers. TAMBI uses public key-based credentials to establish application authorization. When an application makes a call to the WebSphere MQ interface to **put** a message in a queue, the call gets intercepted by TAMBI. TAMBI analyses the call to verify whether this specific application is authorized to do a **put** to the requested queue. If not authorized, it doesn't allow the call to proceed further.

Data protection

If the application *is* authorized to do a **put**, TAMBI proceeds to determine whether the message should be:

- Passed on unchanged
- Digitally signed
- Signed and encrypted.

This decision is based on what has been set as the data protection policy. The options allowed are:

1 No data protection required

2 Message signing only

3 Message signing and encryption.

TAMBI uses the application's private key to digitally sign the message data, which allows later verification that the message has not been tampered with while being processed by WebSphere MQ (both while in a queue and while in transmission to a destination server). Attaching digital signatures to individual messages also allows messages to be traced to their point of origin.

Message encapsulation is performed using PKCS #7 standard and the message data can be unwrapped only by a process that has access to the application’s private key. The encryption strength (RC2 128, DES 64, Triple DES 128, AES 128, or AES 256) can be chosen during policy definition, depending on the specific security need.

Message-level audit functions

TAMBI also provides a message-level audit function, and generates audit records that can demonstrate specific compliance with the defined security policy.

Remote administration of security policies

Since this application’s data protection is handled externally and managed remotely by TAMBI, a single security management solution can be used to protect the messages even though the messages can traverse both mainframe and distributed servers.

CONCLUSION

Because the messages can now be protected, even while they are in the queue, the exposure of sensitive data even to internal employees can be controlled. With the growing data privacy and protection requirements of HIPAA, SOX, etc, this enables the existing applications to be made compliant in a non-intrusive fashion. If Tivoli is used at the Enterprise level, security management of the messaging applications can also be integrated seamlessly.

Sasirekha Cota
Tata Consultancy Services (India)

Prevent queues from jamming your queue manager

INTRODUCTION

One of the more awkward situations you might run into when administering MQ is your queue manager data volume becoming full. If a receiving application, for whatever reason, stops getting messages from its queue, the queue might accumulate messages until your disk volume is completely filled up – a very delicate situation because the queue manager will then be blocked. It refuses even to get messages from its queues! The only way to get it operational again would be to free at least several hundred kilobytes of disk space.

In order to prevent a situation like this, MQ allows the setting of an individual threshold for the maximum number of messages that may accumulate in each queue. A concept of event messages generated by the queue manager in various situations allows, for example, the queue manager to alert you even before the queue fills up. These mechanisms are outlined briefly in what follows. The weaknesses of this concept are pointed out and a tool to cope with them is presented.

MQ STANDARD MECHANISM AND ITS DISADVANTAGES

If the filesystem containing your queue manager data should run full, your queue manager blocks, and you have no chance to get it running again unless you manage to free at least several hundred kilobytes. In order to prevent this situation, MQ allows the setting of a threshold for the maximum number of messages that may accumulate in each queue: the attribute MAXDEPTH of a local queue. If a local queue reaches the threshold, any future MQPUT will fail with a reason code 2053 (MQRC_Q_FULL).

If a completely filled queue is being fed from a remote queue manager, the receiving channel agent consequently cannot deliver any more messages to this queue. Since the channel processes the messages first in first out, it would now be blocked unless you have a dead letter queue defined on the receiving queue manager. The dead letter queue – if configured in the queue manager's properties – now receives the messages as dead letter messages that might easily be re-delivered to the failing queue at a later time. Unless the dead letter queue runs full as well, the channel may still correctly deliver messages to other receivers of this queue manager.

In order to alert you well before any queue runs full, you may activate a queue depth high event on any queue using the queue attributes QDPHIEV, QDPLOEV, QDEPHI, and QDEP-LO. When a queue reaches the high-water mark QDEPHI (by default set at 80%), the queue manager – if you set QDPHIEV(ON) – puts an event message to an event queue (SYSTEM.ADMIN.QMGR.EVENT). This event message can be processed in order to alert you by mail or by a monitoring application such as, for example, Tivoli. If correctly configured, in a similar way, the queue manager sends an event message when the queue depth reaches the low-water mark QDEPLO (default is 20%).

What do you do if you don't have an infrastructure large enough to justify an expensive monitoring tool such as Tivoli? What if you don't have available the skills or the resources to develop a monitoring application that would process the event messages and, for example, send you an e-mail?

The concept of the maximum depth of queues has one great disadvantage – it is entirely based on the number of messages. Hence you need to be able to estimate the size of a message at the time of configuration. If the messages at run-time differ from your estimate, your configuration is inappropriate. If your application happens to receive messages of various sizes – very large ones as well as very small ones – setting a high-water mark based on the number of messages might not be suitable at all. Suppose you set the maximum number of messages based on the size of the largest messages, your queue might run full much too early, when the queue contains an increasing number of small messages. But if you instead set the size of the queue based on the smaller messages received, your disk volume might run full nevertheless, even though you limited the number of messages on the queue.

THE DISK FULL SITUATION AND MAKESHIFTS TO COPE WITH IT

It is very important to prevent the disk volume containing your queue manager data from running full. As pointed out above, your queue manager would be blocked. When a queue manager cannot write its messages, and especially not log data any more, it will still appear to be running. However, it will not be truly operational. Even in order to remove a persistent message from a queue, it would have to be able to write information to the log before destroying the message.

This redundant storage mechanism is vitally important for recovery from a failure, and is very similar to a database. If the queue manager could not document the destructive get operation in its logs, its data would become inconsistent, and the queue manager would not be able to reconstruct its queue data after a fatal termination. Hence the queue manager will not accept any commands until you manage to free at least several hundred kilobytes on the filesystem.

In order to make sure you remain capable of action in an emergency, you could write a large dummy file (eg 1MB or even several megabytes in size) to the queue manager volume, which you could delete if the disk volume should run full. However, MQ delivers messages very quickly. You would have to make sure that the queue manager does not receive so much data within seconds that it fills up your disk again before you've had a chance to intervene.

DISK SPACE MANAGEMENT BASED ON THE PHYSICAL SIZE

A different way to cope with your disk volume running full would be to automatically set a queue PUT(DISABLED) before it fills up your disk volume. Applications trying to put more messages now would fail with a reason code 2051 (MQRC_PUT_INHIBITED) instead of 2053 (MQRC_Q_FULL). The result for the writing application, logically, is equivalent – it fails to put messages to the queue. However, your queue manager now remains operational and you have a chance to cope with the situation before it's definitely too late. Any other queue for this queue manager remains operational and may continue to receive and deliver messages (as long as it does not run full).

In what follows, a concept of MQ disk space management is discussed, which is based on the physical size of the queue file as compared with the remaining space available on the disk volume containing the queue data.

MQ (on non-z/OS systems) stores the message data of a particular queue in a random access file located at:

```
<MQ root>/qmgrs/<qmgr name>/queues/<queue name>/q
```

where:

- *<MQ root>* – Websphere MQ data directory of your installation, on Unix, eg */var/mqm*.
- *<qmgr name>* – your queue manager's name.
- <queue name> – your queue's name (some special characters translated in order make the queue name compatible with file names).

In the following, 100% denotes the current number of bytes available on the disk volume hosting the queue file, provided this particular queue file was not there, ie:

```
fsize + free = 100%
```

where:

- *fsize* – number of bytes currently occupied by this particular queue file

- *free* – number of bytes currently available on the disk volume hosting the queue file.

Similarly (but opposed) to the concept of queue depth high and queue depth low events of MQ, a relative headroom is defined:

```
hr  =  100%  -  fsize / (fsize + free)
```

When queues are monitored, the condition for regular operation then is:

```
hr  ≥  hrlo
```

Should a queue's fsize increase such that its headroom drops below this hrlo limit, it will be disabled for put and a warning will be issued. When its size decreases again, such that its headroom exceeds the hrhi limit again, ie:

```
hr  ≥  hrhi
```

the queue might automatically be re-enabled for put, while again sending a warning to the MQ administrator.

DISK SPACE MANAGEMENT PROGRAM FOR UNIX OR WINDOWS

The Perl program amqfsmon.pl uses the concept addressed in the previous section. The monitoring of disk space usage at present is supported for Unix and Windows (Win32) systems. Unix systems generally contain a Perl interpreter in their original distribution. In order to operate amqfsmon.pl on Windows systems, however, a Perl interpreter must be installed (eg ActiveState ActivePerl, freely downloadable, currently V5.8). For other types of operating system, amqfsmon.pl could easily be extended by the user.

The utility does not depend on any MQ for Perl module. The MQ interactions are programmed by the aid of calls to MQ control commands such as **runmqsc** or **dspmqfls**. Hence a plain, regular Perl distribution is sufficient.

The program amqfsmon.pl is called as follows:

```
amqfsmon.pl  -l hrlo  [ -m qmgr ]  [ -h hrhi ]  [ -w wait ]
[ -n ]  [ -r rcpt [ , rcpt ... ] ]  [ -s svr ]  [ -x ]
[ -v ]
```

where

- -l *hrlo* – relative headroom [%] of queues to be kept free (required).
- -m *qmgr* – queue manager to be monitored (default = MQ default queue manager).
- -h *hrhi* – relative headroom [%] of queues, threshold at which queues previously disabled are re-enabled (default = hrlo).
- -r *rcpt* – mail recipient(s) to report warnings (default = none).
- -s *svr* – DNS name of the mail (SMTP) server (default = localhost).
- -n – never re-enable queues (optional; the queues must be re-enabled manually).
- -v – (verbose) in addition to the logging and the (optional) mailing by SMTP, write messages and warnings to standard out (optional).
- -w *wait* – seconds to wait between headroom checks (default = 600).
- -x – monitor transmission queues only (default = all queues).

The utility monitors the space occupied in the filesystem by each queue of the queue manager qmgr (transmission queues only if flagged by **-x**; see below). It periodically polls all local queues at the interval specified by the flag **-w** and sets a queue PUT(DISABLED) if its relative headroom decreases below the threshold hrlo. When the headroom exceeds hrhi ≥ hrlo again, the queue is re-enabled (unless flagged by **-n**).

If a local default queue manager is defined, the only parameter required is the lower threshold, hrlo. If hrhi is omitted, it is assumed to be equal to hrlo. It is, however, recommended to change hrhi to a value >hrlo, for example hrhi = 20% and hrlo = 10%. These definitions will keep an individual relative headroom of at least 10% for each queue. Once a queue reaches the low limit, hrlo, it is automatically re-enabled for put, but not before the headroom exceeds the high limit, hrhi, again.

Messages about the disabling and the re-enabling of queues are written to the file *amqfsm01.log* in the errors directory of the queue manager. Three log files are maintained. Once it reaches 256KB, *amqfsm01.log* becomes *amqfslog02.log* and then *amqfslog03.log* and the primary log file, *amqfslog01.log*, is re-allocated.

If the optional flag **-r rcpt** is specified, the warnings are sent to the mail recipient(s) **rcpt** (comma-separated list of addresses). If the local host does not provide a mail (SMTP) server, a different server IP address or DNS name, **svr**, needs to be specified with the aid of the (optional) flag **-s**.

The flag **-v** sends all warnings and messages to standard output as well to the log and the mail receivers. It was introduced for analysis and configuration purposes. For additional safety, the flag **-n** may be used. It prevents automatically disabled queues from being automatically re-enabled. If the flag **-n** is specified, the MQ administrator must manually re-enable any queues that were disabled for put.

The flag **-x** is designed to monitor just the outgoing queue connections of a queue manager. The transmission queues, exclusively monitored by specifying **-x**, receive any messages that are sent to remote queues. They are the (last and only) local physical queues that accept these messages before they are transmitted across the channel to the remote queue manager.

CONCLUSIONS

The problems with queue manager data volumes running full have been addressed and weaknesses have been identified in the IBM concepts of queue depth limitation and event generation. Methods to cope with them have been presented. The script published with this article allows you to manage the occupation of queue manager data volumes based on the physical space used and available. Using the methods and the tool presented, an administrator should always be in a position to successfully operate their MQ environment and cope with the possibility of a queue filling up the filesystem.

AMQFSMON.PL

```
#!/usr/local/bin/perl
#
# Name:    amqfsmon.pl  (perl 5.Ø script)
#
# Version: 1.1.Ø
#
# Author:  Dr. Johannes Boehm, ISDD
#          Giessenstrasse 11, CH-86Ø8 Bubikon, Switzerland
#          mailto: j.boehm@gmx.ch
#
# Purpose: Monitors the sizes of queue files in the filesystem and sets
#          a queue PUT(DISABLED) if its headroom decreases below a user-
#          defined threshold. Further details see help string below.

use diagnostics;
use warnings;
use strict;

use Sys::Hostname;
use POSIX;
use Net::SMTP;

$SIG{__DIE__}  = \&errorExit;   # set up simple trap
sub errorExit { print shift; exit 1 }

my $nl="\n          ";   # new line including indent of 1Ø spaces

my $help="\n" .
"Usage:    amqfsmon.pl  -m <qmgr>  -l <hrlo>  [ -h <hrhi> ]  [ -w <wait>
]" . $nl .
"   [ -r <rcpt> [ ,<rcpt>... ] ]  [ -s <svr> ]  [ -n ]  [ -x ]  [ -v
]\n\n" .
"Options:  -l <hrlo> Relative headroom [%] of queues to be kept free
(required)" . $nl .
"-h <hrhi> Relative read room [%] of queues, threshold at which queues"
. $nl .
"          previously disabled are re-enabled (default = <hrlo>)"
. $nl .
"-m <qmgr> Queue manager to be monitored (default = MQSeries default)"
. $nl .
"-r <rcpt> mail recipient(s) to report warnings (default = none)"
. $nl .
"-s <svr>  dns name of the mail (smtp) server (default = localhost)"
. $nl .
"-n        Never re-enable queues (they must be re-enabled manually)"
. $nl .
"-v        Verbose, write messages and warnings to stdout (optional)"
. $nl .
"-w <wait> Seconds to wait between size checks (default = 600)"
. $nl .
"-x        Monitor transmission queues only (default = all queues)" .
"\n\n" .
"Synopsis: " .
"Monitors the space occupied in the filesystem by each (transmission)" .
$nl .
"queue of the queue manager <qmgr> and sets the queue to PUT(DISABLED)"
. $nl .
"if its relative headroom decreases below the threshold <hrlo>.  This" .
$nl .
"headroom is defined as  100 * fr / (sz + fr)  where 'sz' denotes the" .
$nl .
"space in kB occupied by the queue  and 'fr' the space left.  When the"
. $nl .
"headroom exceeds  <hrhi> >= <hrlo>  again,  the queue  is re-enabled" .
$nl .
"(unless flagged by -n)." . $nl .
"Messages about  the enabling and the disabling of queues  are written"
. $nl .
"to the file amqfsm01.log in the errors directory of the queue manager"
. $nl .
"and, if specified,  to the mail recipient(s) <rcpt>  (comma separated"
. $nl .
"list of addresses).  If the local host does not provide a mail (smtp)"
. $nl .
"server,  a different server <svr>  may be specified.  Three log files"
. $nl .
"are maintained.  At a size  of 256 KB amqfsm01.log the  log files" .
$nl .
"are moved down  to amqfslog02.log and amqfslog03.log  and the primary"
. $nl .
```

```
"log file amqfslogØ1.log is re-allocated.\n";

my %arg = (      # preset argument defaults:
   m => '',      # queue manager name = empty (required)
   l => '',      # headroom low = empty (required)
   h => '',      # headroom high (default = headroom low, see getArgs)
   n => Ø,        # noEnable = off
   r => '',        # mail address(es) = empty
   s => hostname(), # mail (smtp) server, default = localhost
   v => Ø,           # verbose = off
   w => 6ØØ,          # wait = 6ØØ seconds
   x => 'NORMAL|XMITQ' # monitor all types of queues
);
%arg = getArgs(@ARGV);  # get cmd line args/options to global hash %arg

if ($arg{p}) {           # if internal flag -p present (see sub runmqsc)
    print "$arg{p}";     # then simply print the argument
    exit Ø;              # and exit without error return code
}

my %qs;     # declare global hash of queue names/attributes
my $logf = getQmDir($arg{m});   # get log file path (qmgr directory)

while ( 1 ) {                              # do forever (until interrupt)
  getQattr('usage', '*', $arg{m}, \%qs);  # get q usage for all qs

  if ($arg{v}) {
    my $noqs = scalar keys %qs;
    print (
      strftime "%d.%b.%Y, %H:%M:%S, qmgr $arg{m} ($noqs queues):\n",
      localtime
    );
  }

  foreach ( keys %qs ) {         # for all queues found in qmgr $arg{m}
    checkQ($_, \%arg, \%qs)           # check the q size if q usage
      if $qs{$_}{usage} =~ /$arg{x}/; #   found in usage from the cli
  }

  sleep $arg{w};  # wait for the interval received from the command line
} # end: while forever


################
# Subroutines #
################

sub checkQ  # purpose:   Checks the size of the q file & writes a warn-
##########  #            ing to the log if the headroom drops below the
            #            threshold $arg{l}.
            #            Uses predef. string $logf (log files).
            # arguments: 1. queue name
            #            2. reference to hash of arguments
            #            3. reference to hash of queues/attributes
{
  my ($q, $arg, $qs) = @_;   # get q name & args hash & q hash refs.
  my @qf = grep(/queues/i, `dspmqfls -m $$arg{m} -t q $q`);
                             # get q file path from dspmqfls
  my $qf = pop @qf;          # current q directory = last (only) entry
  chomp $qf;

  if ( -d $qf ) {                  # if queue directory exists

    if ( $^O eq 'MSWin32' ) {        # if OS = Windows
      my $free = `dir "$qf"`;          # get free space at q directory
      $free =~ s/^.*\s(\S+)\s+bytes\s+free/$1/s;
                                       # extract value 'bytes free'
      $free =~ s/\D//g;                # remove group separators
      $$qs{$q}{free} = $free / 1024;   # bytes -> kB
      $qf .= '\\Q';                    # append file name to directory
    } else {   # Unix
      $$qs{$q}{free} = `df -b $qf |    # get free filesystem kB at q
        tail -1 | awk '{print \$2}'`;  #   file & extract size (2nd col)
      chomp $$qs{$q}{free};
      $qf .= '/q';                     # append file name to directory
    } # end if $^O

    $$qs{$q}{fsize} = (stat $qf) [7];  # get q file size in bytes
    $$qs{$q}{fsize} /= 1024;           # queue file size, bytes -> kB
    $$qs{$q}{hroom} = $$qs{$q}{free} * 100 /   # headroom
      ($$qs{$q}{fsize} + $$qs{$q}{free});
    $$arg{v} and print "$q: $$qs{$q}{fsize} kB, " .    # verbose ->
      "$$qs{$q}{free} kB free ($$qs{$q}{hroom}%).\n";  #   print comment

    if ( $$qs{$q}{hroom} < $$arg{l} ) {   # if rel.headroom < threshold
      disableQ($q, $arg, $qs);            #   => put disable the queue
    } else {                      # else, if rel.headroom >= hi thresh.
      reEnableQ($q, $arg, $qs);   #   => re-enable q or warn if exceeded
    } # end if headroom < threshold

  } else {
    $$arg{v} and print "$q: No file $qf found.\n";
  } # end if q dir exists

} # end: checkQ ========================================================



sub disableQ  # purpose:   Set a queue to PUT(DISABLED) and warn
############  # arguments: 1. queue name
```

```
             #            2. reference to hash of arguments
             #            3. reference to hash of queues/attributes
{
  my ($q, $arg, $qs) = @_;                  # get arguments
  getQattr('put', $q, $$arg{m}, $qs);      # get put attr. of curr. q

  if ( $qs{$q}{put} eq 'ENABLED' ) {       # if q still enabled, then
    my $cmd = "alt ql($q) put(disabled)"; #   mqsc to put-disable the q
    my ($rc) = runmqsc($cmd, $$arg{m}, "^AMQ");
                                          #   run cmd, keep MQRC only
    chomp $rc;
    $$qs{$q}{disabl} = 1;                 #   keep q disabled indicator
    warning("disabled for put", $rc, $q, $arg, $qs);
                                          # write warning to log & mail
  } # end if enabled

} # end: disableQ =====================================================



sub reEnableQ  # purpose:   Re-enable a queue if its high threshold is
#############  #            exceeded again and warn if not.
               # arguments: 1. queue name
               #            2. reference to hash of arguments
               #            3. reference to hash of queues/attributes
{
  my ($q, $arg, $qs) = @_;

  if [ $hr -lt $hrhi ]; then                # if rel. headroom < thresh.
lo
    [ $hr -lt ${lsthr[$2]} ] &&             # relative headroom
decreasing
      warn "filling up the filesystem!"   #   -> issue a warning
  elif [ ${disabl[$2]} -eq 1 ]; then       # if $hr >= thresh. & q was
disabled

  if ( $$qs{$q}{hroom} > $$arg{h} ) {  # if rel. headroom > thresh. hi
        if ( $$qs{$q}{disabl} ) {           # if q is disabled
      $$qs{$q}{disabl} = Ø;            # reset q disabled indicator
      $$arg{n} and return;             # re-enable flagged off -> return
      my $cmd = "alt ql($q) put(enabled)";   # mqsc to set q put enabled
      my ($rc) = runmqsc($cmd, $$arg{m}, "^AMQ");   # run cmd, keep MQRC
      chomp $rc;
      warning("re-enabled for put", $rc, $q, $arg, $qs);
                                       # write warning to log & mail
    } # end if q is disabled
  } else {
        warning('filling up the filesystem!', 'none', $q, $arg, $qs);
  } # end if rel. headroom > threshold high
} # end: reEnableQ ====================================================


sub getQattr  # purpose:   Get an attribute of a queue
############  # arguments: 1. attribute name
              #            2. queue name
              #            3. queue manager name
              #            4. reference to hash of queues/attributes
{
  my ($a, $q, $qm, $qs) = @_;  # get q attr & name, qmgr, & q hash ref.
  my ($k, $v);                 # local key/value pair for return hash

  foreach ( runmqsc("dis ql($q) $a", $qm) ) {  # display attr $a of q $q
    chomp;
    ($k=$_) =~ s/^.*QUEUE\(([^)]+).*$/$1/   # extract q name as key
      if /QUEUE\(/;                         #   if 'QUEUE' found

    ($v=$_) =~ s/^.*$a\(([^)]+).*$/$1/i     # extract attribute as value
      if /$a\(/i;                           #   if attribute found

    if ($k and $v ne '') {                  # if both key & val. defined
      $$qs{$k}{$a} = $v;                    #   then assign to q hash
      $k = $v = '';                         #   and flush key/value
    } # end if

  } # end for

} # end: getQattr =====================================================


sub getQmDir  # purpose:  Extract properties of the current queue
############  #           manager from mqs.ini and set mq attributes
              #           used by setQs()/checkQ().
              # argument: queue manager name
              # returns:  $logf<n> (path to log files 1/2/3 of amqfsmon)
{
    my $qm = shift;

    ($_) = runmqsc('ping qmgr', $qm, 'AMQ8146|AMQ8118');
                                            # check if qmgr is running
    s/manager/manager '$arg{m}'/;           # insert qmgr name into msg
    die $_ if $_;                           # die if not running
    ($_) = grep /qmanager.qmanager/i, 'dspmqfls -m $qm qmgr';
                                            # get mq path by dspmqfls
    chomp;
    s/qmanager/errors/i;    # repl. 1st occ. of 'qmanager' by errors dir
    s/qmanager/amqfsmon/i;  # repl. 2nd occ. of 'qmanager' by log name
    return $_;              # return resulting log file path
```

```
} # end: getQmDir =====================================================


sub runmqsc  # purpose:   Run mqsc commands by runmqsc and return their
###########  #            output.
             # arguments: 1. mqsc command(s)
             #            2. queue manager name
             #            3. grep pattern for filtering (optional)
{
  my ($cmd, $qm, $grp) = @_;              # get arguments
  my @out = '$Ø -p "$cmd" | runmqsc $qm'; # echo $Ø to pipe -> OS-in-
                                          #   dependent, no double pipes
  @out = grep(/$grp/, @out) if $grp;      # filter lines if required
  return @out if @out;    # return @out only if its not empty
  return ('');            # return array of 1 empty string if @out undef
} # end: runmqsc ======================================================


sub warning  # purpose:  Write a warning to the log file, the mail reci-
             #           pient, and stdout
###########  # argument: 1. Message to display (use other vars from
             #              checkQ below without passing them)
             #           2. MQ reason code from previous mqsc
             #           3. Current queue name
             #           4. Reference to argument hash
             #           5. Reference to queue hash
{
  my ($msg, $rc, $q, $arg, $qs) = @_;
  my $logfs = (stat "${logf}Ø1.log") [7] or Ø;  # get log file size

  if ( $logfs > 262144 ) {               # if log file exists & > 256 kB
    rename "${logf}Ø2.log", "${logf}Ø3.log";  # move log file 2 to 3
    rename "${logf}Ø1.log", "${logf}Ø2.log";  # move log file 1 to 2
  } # if log file too large

  getQattr('curdepth', $q, $$arg{m}, $qs);  # get depth of current queue
  my $host = hostname();

  my $out =                                    # set up message
    strftime("%d.%b.%Y, %H:%M:%S", localtime) . #   with date/time,
    ", qmgr $$arg{m} on host $host:\n" .        #   qmgr name/host name,
    "   Queue '$q' $msg (rc=$rc)\n" .           #   warning with q name,
    "   File size   = $$qs{$q}{fsize} kB,\n" .  #   current file size,
    "   Queue depth = $$qs{$q}{curdepth} messages,\n" . # q depth, &
    "   Headroom   = $$qs{$q}{free} kB / " .           # headroom info
      "$$qs{$q}{hroom}% left, $$arg{l}% required.\n";
  open (LOG, ">>${logf}Ø1.log");
  print LOG $out;         # write message to log
  close LOG;

  if ($$arg{r}) {  # recipient ok -> send mail
    my $smtp = Net::SMTP->new($$arg{s}); # open socket to smtp server
    $smtp->mail("amqfsmon\@$host");     # initiate sending of a message
                                        #   by passing sender address
    $smtp->to($$arg{r});                # notify server about recipients
    $smtp->data();                      # initiate sending of the data
    $smtp->datasend("To: $$arg{r}\n");  # send data: receiver identific.
    $smtp->datasend("Subject: Warning: Queue $q $msg!\n");
                                        # send subject specification
    $smtp->datasend($out);              # send message
    $smtp->dataend();                   # end sending data
    $smtp->quit;                        # close socket connection
  }

  print $out if $$arg{v};  # verbose -> msg to STDOUT too

} # end: warning =====================================================


sub getArgs  # purpose:   Get the arguments/flags from the command line.
###########  # arguments: none
             # returns:   hash of all flags
{
  $_ = join(" ", @_);                # get all cmd line args
  my $flg = 'hlmnprsvwx';            # set of all valid flags
  my $sfl = 'nvx';                   # subset of valid simple flags

  /-[$sfl]*([^$flg ])/ and           # if an invalid flag is encountered
    die "\nInvalid option: -$1.\n$help";  # die with help

  while (s/-([$sfl])([$flg])/-$1 -$2/) {} # separate simple flags
  s/(^|\s+)-([$sfl])/$1-$2 1/g;           # add value 1 to simple flags
  s/(^|\s+)-([$flg])\s*/\e$2\e/g; # '-<flag> <val>' -> '\e<flag>\e<val>'
                                  #    and add ' ' for simple flags
  s/x\e1/x\eXMITQ/g;              # set value for simple flag: -x
  s/^\e//;                        # remove leading delimiter

  %arg = (%arg, split "\e");      # overwr. defaults hash by avail.args

  if (not $arg{p}) {   # if internal print flag is not set, then
        / (\S+)/ and die "\nCannot interpret argument: $1.\n$help";
                                   # die if stand-alone argument found
    die "\nRequired argument missing: <hrlo>.\n$help" unless $arg{l};
                                  # die if required flag -l missing
    die "\nRequired argument missing: <qmgr>.\n$help" unless $arg{m};
```

```
                                  # die if required flag -m missing
    die "\nArgument <hrlo> must be numeric.\n$help"
      unless $arg{l} =~ /\d+/;    # die if <headroom lo> not numeric
    die "\nArgument <wait> must be numeric.\n$help"
      unless $arg{w} =~ /\d+/;    # die if <wait> not numeric
  }

  $arg{h} = $arg{l} unless $arg{h};  # dflt. headroom hi = headroom lo
  return %arg;

} # end: getArgs =====================================================
```

Dr sc nat Johannes P Boehm
IBM certified MQSeries Specialist,
IBM certified MQSeries Developer,
IBM certified MQSeries Solutions Expert (Switzerland)

Contributing to *MQ Update*

Why not share your expertise and earn money at the same time? *MQ Update* is looking for program code, scripts, REXX EXECs, JavaScript, etc, that experienced users of MQ have written to make their life, or the lives of their users, easier. We are also looking for explanatory articles, and hints and tips, from experienced users. We would also like suggestions on how to improve MQ performance.

We will publish your article (after vetting by our expert panel) and send you a cheque, as payment, and two copies of the issue containing the article once it has been published. Articles can be of any length and should be e-mailed to the editor, Trevor Eddolls, at trevore@xephon.com.

A free copy of our *Notes for Contributors*, which includes information about payment rates, is available from our Web site at www.xephon.com/nfc.

WAS 6 and autonomic computing

IBM announced WebSphere Application Server Version 6 in October last year. It contains support for various Java and Web services standards – the Web services specification, WS-I Basic Profile 1.1, J2EE 1.4, the Java API for building Web services and clients using RPC and XML, and JAX-RPC. It also included improvements in autonomic computing for WebSphere.

In this article I will look first at some of the more significant announcements and then at why autonomic computing is important for a company.

SIGNIFICANT ANNOUNCEMENTS

There is now greater consistency in the WAS family from top to bottom, including in the Express version. Gone is the need for customized builds of the Application Server for WAS Express, Base, Network Deployment, and Enterprise. WebSphere 6 Express/Base Single Server CDs now provide one set of images that install Express or a Single Base Server, and offer full J2EE 1.4.2 support for both versions, which makes it easier to develop and deploy applications using industry-standard tools. Also included with the Express/Base product are some features that were previously only available in the Enterprise Edition – including Workmanager (previously known as Asynchronous Beans) and Application Profiles.

The inclusion of better scaling means that more users can access an application simultaneously. This will, therefore, provide better flexibility, and means that licensing and administrative costs will be reduced.

Versions 5.0 and 5.1 allowed the creation of multiple instances of WAS using the **wsinstance** command. With Version 6 this is enhanced and called Server Profiles.

The Web services standards have been updated and there are some additional features. Web services standards are more automated, which provides cross-platform computing. WS-I Basic Profile 1.1, WS-Transactions, and WS-Security are all supported. A standards-based approach allows companies to respond efficiently and speedily to any changes in their business, and this can all be incorporated into their Service-Oriented Architecture (SOA) strategy.

SOA allows a company to interconnect business functions, processes, and services using industry-standard software components rather than manually-coded ones. It allows a huge flexibility in the combinations that are possible.

Associated with SOA is an Enterprise Service Bus (ESB). This is basically a connection infrastructure allowing business transactions to flow from one application to the next. WAS Version 6 makes it simpler to connect WebSphere applications to an ESB. It now has a faster messaging capability and connects easily with the IT infrastructure that already exists.

Service Data Objects (SDO) simplify and unify the way that applications handle data. Using SDO, application programmers can uniformly access and manipulate data from heterogeneous data sources, including relational databases, XML data sources, Web services, and enterprise information systems.

Java Server Faces (JSF) is a component-based framework bringing user interface development methods for building Java fat client user interfaces to Java-based Web application development.

The ASTK (Application Server Toolkit) has been updated in Version 6. The ASTK provides WebSphere developers and administrators with a set of Eclipse-based tools for assembling, deploying, debugging, and profiling J2EE applications.

For developers there are a number of new wizards and a drag-and-drop environment. This automates what can be routine and tedious steps in the development and deployment of an application. Eliminating hand coding reduces the number of programming steps needed to build an application. WebSphere Rapid Deployment has improvements in annotation-based programming, deployment automation, and change-triggered processing.

AUTONOMIC COMPUTING

WebSphere Version 6 is designed to detect problems automatically. It can also save and process Web-based business transactions automatically. Previously, this could have been a long process.

IBM has produced a lot of figures explaining how much downtime costs, quoting averages of $6.5 million per hour in the retail brokerage industry, $2.6 million in credit card authorizations, $90,000 in airline reservation centres, $27,000 in manufacturing, and $17,000 in banking.

Autonomic self-healing software is a route that IBM started to go down with DB2 a while ago, and has continued development in with its recent DB2 announcements.

WAS Version 6 is able to configure, optimize, heal, and protect itself. IBM explains the need for autonomic software by saying that the management of systems is getting so complex, it is easier to let the software do it and let companies get on with what they know best – running their businesses.

When it detects an outage, WAS Version 6 redirects data to a different designated ‘failover’ server automatically. This second server will usually be within the same data centre, but, should the failure be more serious in nature (like a local disaster!), WAS can move the information via the Internet to a completely different location.

Nick Nourse
Independent Consultant (UK)

WebSphere MQ on high-capacity Unix servers: access control

ABSTRACT

This article is about controlling access to WebSphere MQ administration facilities in a multi-user Unix environment. Subject to the local IT policies, this task may pose some operational challenges, solutions for which are not readily available in WebSphere MQ or third-party products known to me. The article describes a simple and practical solution to those problems.

IBM WebSphere MQ (formerly known as MQSeries) allows users easily to exchange information across different platforms, integrating new and existing business applications. Further in this article, we will refer to it just as MQ.

PROBLEM

Often, organizations choose to host a lot of queue managers on a high-capacity Unix server. Such a strategy, commonly known as server consolidation, allows companies to reduce some costs, including MQ licensing costs, better leverage the purchased computing power, and save on Unix system administration and some other data centre costs.

In such an environment, it is common to find queue managers belonging to distinct application support groups (hereinafter called simply application groups), often from different departments or business lines. It is often the case that, before migrating their queue managers into the consolidated environment, many of these application groups managed their queue managers using their internal resources. For a number of more or less valid reasons the application groups may prefer to keep administrative access to their queue managers rather than organize a dedicated MQ administration group and surrender their access to this group.

Among the valid reasons are the significant differences in the support levels expected by different groups, the high cost of the dedicated 24x7 centralized MQ administration group, and the large amount of MQ expertise that can be accumulated by the application groups, who can integrate MQ support with their other responsibilities.

As soon as it is decided that the application groups will continue to manage their queue managers, they discover the operational risk caused by the absence of the necessary access control safeguards that prevent one group from affecting other groups' queue managers. To see the problem more clearly, let us list some facts about MQ system administration techniques on Unix:

1 The main means of MQ administration on Unix platforms is so-called control commands, which are simply Unix executables. Some operations, like queue manager creation or deletion, can be performed only by control commands.

2 The access to MQ control commands is managed by means of the Unix operating system. Specifically, in order to run any control command, the user of the calling process must be allowed in Unix group mqm.

3 Anyone permitted to run a control command can apply it to any queue manager on the machine. There is no documented way to restrict the access of one person who is allowed to run control commands on a given list of queue managers.

To summarize, the standard MQ installation on Unix does not have any way of restricting administrative privileges to a specific set of queue managers, and this becomes a problem for many IT departments.

THE IDEA BEHIND THE SOLUTION

Before any further discussion, it is essential to know that every time a control command is executed, it operates on one, and only one, queue manager. The name of this queue manager is always specified by one of the control command arguments. For example, the following is the generalized syntax of the control command creating a new queue manager:

```
$ crtmqm [<options>] <queue-manager-name>
```

Other commands have a similar syntax. There are slight differences in where the *<queue-manager-name>* is to be specified in the command line and whether it must be given as the value of an option (usually, **-m**) or just in a file argument style, as in **crtmqm**.

(Strictly speaking, it is possible to designate a queue manager 'default', in which case it is possible to omit its name from the control command calls. The concept of the default queue manager is rather more harmful than useful in the environments we are interested in [with multiple queue managers belonging to independent owners it is clearly problematic to nominate any particular one as more equal than others]. From further discussion it will become obvious that our approach prevents default queue managers from ever being created.)

Given that, one could think of constraining the control command syntax for every member of an application group with the requirement to use only the names of the queue managers owned by their application group. Such a requirement is not directly enforceable in Unix, but the idea is still fruitful and here is one approach that works:

1 Bar any application group from executing control commands directly.

2 Allow any application owner to call control commands via some trusted agent that will pass only the command lines referring to the owner's queue managers. Of course, the

agent must allow its user to enter any useful combination of other control command argument.

An additional requirement of the agent is that it (and its configuration, if any) must be protected from being tampered with by application group members.

Non-functional requirements:

1 Ease of use. This includes a minimal possible learning curve for seasoned MQ administrators (ideally, a zero learning curve).

2 Minimal burden and learning curve for Unix SAs to set up the agent.

3 Low total cost of ownership.

4 Low implementation cost.

SOLUTION

Here is the precise architecture of the solution I implemented based on the principles above:

1 Unix SAs do not allow any application owner in the group mqm.

2 Unix SAs designate a separate group for every subset of queue managers owned by a distinct application group, for example 'agroup'.

3 For every new queue manager on the system, Unix SAs generate a set of shell scripts with names that resemble control commands. Each script 'wraps' the call to its respective control command. Before executing the control command, the script validates its arguments using the syntax of the control command documented by IBM. Additionally, the script validates that the queue manager name is present and is equal to the script's target queue manager name. If any of the validations fails, the script

exits without executing the command; otherwise, the command is executed.

4 The system must have the open source tool **sudo** installed. **Sudo** is used to let an ‘agroup’ member execute the control command with the credentials of user mqm. In order to avoid introducing application groups to the **sudo** command and make their work as similar to the regular MQ administration as possible, the wrapper scripts recursively call themselves with **sudo**, like this:

```
test "X$SUDO_COMMAND" = "X$cmd" || exec /usr/bin/sudo -u cmd $cmd
```

5 For every new queue manager on the system, Unix SAs add the fragment to the */etc/sudoers* file, similar to the following:

```
# This section is for 'groupa' to administer queue manager QM.A
# (MQ core control commands)
%groupa myhost = (mqm) NOPASSWD: /usr/mqm/wrappers/QM.A/w_crtmqm
%groupa myhost = (mqm) NOPASSWD: /usr/mqm/wrappers/QM.A/w_dltmqm
...
```

This example refers to the standard MQ installation path on an AIX operating system.

6 The scripts are put in the subdirectory *wrappers/<queue-manager-name>/* of the MQ installation directory. The *wrappers/* directory is owned by user and group mqm and has read and execute permissions only for the user and the group. The *wrappers/<queue-manager-name>/* directory is owned by user mqm and group agroup and has read and execute permissions for the user and the group. It is essential that this directory does not have write permissions for the group.

The scripts, with the names resembling the control command names, like ‘w_crtmqm’, are owned by the user mqm and group agroup and have read and execute permissions for the user and the group. Again, it is essential that the scripts are not writable by the group.

7 The application group members add the subdirectory *wrappers/<queue-manager-name>/* of the MQ installation directory to their system PATH and use **wrappers** as they would use regular MQ control commands, with the following two small differences:

- the name of every control command needs to be prefixed with ‘w_’.
- the control command syntax employing the default queue manager is prohibited.

There are quite a few things to take care of in the wrappers to make sure they cannot be hacked by submitting unusual arguments to them or running them in a ‘poisoned’ environment. Also, quite a bit of work is required from Unix SAs to install wrappers and configure **sudo**. Therefore, I wrote a program in Perl to automate this task. Given the application group name and queue manager name, the program generates and installs the set of wrappers for core MQ control commands or MQ PubSub extension control commands, depending on the user’s choice. Also, the program generates the commented fragment of */etc/sudoers* for granting the group members the **sudo** access to the wrappers with the credentials of mqm user. The current version supports AIX and Solaris operating systems and adding support for another version of Unix should not be difficult.

Some control commands are inherently capable of affecting multiple queue managers with the privileges of mqm, even though they are run for a particular queue manager. These are the commands operating with the MQ trigger monitor and default DLQ handler. To stay on the safe side, the wrappers for these commands are intentionally not generated. In the shared environment, without centralized MQ administration, my recommendation is to make a copy of the respective **runmqtrm** and **runmqdlq** commands without the setuid bit and make them available for any user on the system to run with their own credentials rather than those of mqm.

CONCLUSION

The solution discussed in this article has been used successfully in our organization for more than three years. No real problem has ever been reported and the application owners seem to be happy with it.

Ironically, no commercial package providing the analogous functionality seems to be available. Most of the solutions for MQ administration are based on the command server and PCF commands. These are powerful tools. However, their use requires that at least one queue manager be up and running on the system, which defeats the purpose of decentralizing MQ administration.

You are welcome to send your questions and comments about this article to me at paultolk@yahoo.com.

Pavel Tolkachev
Software Architect (USA)

Defining WebSphere MQ roles using sudo

INTRODUCTION

In most companies certain tasks have to be performed by more than one team. On Unix systems these teams may be Unix groups. In the context of WebSphere MQ, it is quite easy to define several groups with different permissions on WebSphere MQ objects – eg by using the default Object Authority Manager (OAM) of WebSphere MQ. It is more complicated to restrict access to WebSphere MQ programs in a detailed way. Some Unix derivatives provide Access Control Lists (ACL) to enable a more detailed structure than Unix generally has. But such mechanisms are not available on every Unix derivative.

This article describes a solution that makes use of a very popular freeware tool named **sudo**. The name of this tool contains the Unix tool **su** (stands for ‘set user’) and the word ‘do’. This means **sudo** allows users to execute (do) programs under a different user id from the caller – without needing to know the password of this other user. The tool **sudo** maps the user id, checks the permissions of the user, and executes the specified program – if the caller is allowed. In addition, if configured, **sudo** logs the usage of the commands defined in the **sudo** configuration.

THE SUDO TOOL

Reasons for using sudo

There are several reasons to use **sudo** on Unix platforms. It is available on any Unix platform – at least by translating the free sources to your particular system. There is one single configuration file containing all the information about which user or group may run which commands. The tool is very flexible and easy to use. **Sudo** is able to control any program and run it using any user for execution. The only trick is to create a proper configuration file.

This article describes a configuration specific to WebSphere MQ. Nevertheless **sudo** is able to control any other program, like a database or whatever. Because of this, **sudo** is probably available and in use on most Unix systems. In this case, it is very easy to extend it for WebSphere MQ needs; otherwise, it is very easy to install or configure it for WebSphere MQ needs.

Sudo allows users to run programs using other user ids, without needing to have the password for this other user id. Users may execute these programs using their own private password or even without any password – depending on the configuration and your security requirements. It is also possible to enable users for a list of programs, for only a single program, or even for a single program with specific parameters. Execution permissions may be defined in a generic form too.

Because of this flexibility of **sudo**, it is very important to think about the permissions that are needed in your environment before the **sudo** tool is configured. This article describes a solution that should prove very useful for most companies. However, the configuration may need to be adapted for your own needs.

How sudo works

The program **sudo** is a freeware solution that provides standard users with extended privileges in special cases. The tool executes specified programs using a privileged user id – like root or the WebSphere MQ system user mqm. There is no need to know the password of this special user, so users may be added to the group without publication of a password. Also, it is very easy to remove users from these groups without changing the password afterwards. **Sudo** itself runs with root privileges, so it is able to run any other program.

Sudo checks the configuration, defined in the file */etc/sudoers*, to see whether or not an action is allowed for a user. Any actions performed by **sudo** are stored in a log file – if one has been specified. The following line defines the file */var/adm/sudolog*, which is then used by **sudo** to log its actions:

```
Defaults logfile = /var/adm/sudolog
```

If no logfile is specified, the actions are *not* reported. So it makes sense always to have a log file. **Sudo** is able to handle four kinds of alias definition. Two of them are used in this article – Host_Alias and Cmnd_Alias.

Host_Alias is a flag to create a list of host names that have the same **sudo** configuration. If it not necessary to have different configurations on different hosts; it is possible to use the value **ALL** instead of a host list. In this article I use different lists for AIX platforms and other systems (Sun Solaris and Linux) because the program paths differ on these systems.

The second alias, Cmnd_Alias, is used to create several groups of commands, which have to be enabled for a special

Unix group. In fact most of these aliases are defined twice, once with an extension _AIX for AIX systems, and then without this extension for any other platform. The WebSphere MQ sample programs by default are located in different directories (*/usr/mqm/samp/bin* and */opt/mqm/samp/bin*), whereas the other WebSphere MQ tools are linked to the path */usr/bin*. The samples below use the direct paths to the program's directories (directories */usr/mqm/bin* and */opt/mqm/bin*) instead of the links.

SAMPLE CONFIGURATION

Setting up the sudo configuration

Before you start to configure the **sudo** tool, you have to think about your security requirements and about the user groups and what application they should be able to get access to. It is possible to enable specific user names to execute the programs. But in most cases there will be a team that performs specific tasks and all team members need the same permissions. Design your roles and clarify the needed permissions before you set up the **sudo** configuration file.

When the role structure is defined, create a Unix group for each **sudo** role you need. Now create some command aliases containing the programs and tools that have to be enabled for this **sudo** role. If possible, use wildcards for the command aliases (eg by grouping the commands in special directories or by using name spaces). More **sudo** roles means more Unix groups, which means more administration work. Fewer **sudo** roles means fewer Unix groups, which means less granularity in your security configuration. You have to come up with a solution that fits the following maxim:

'As much as necessary, as little as possible.'

Description of the sample sudo configuration

The sample configuration below is divided into four parts. Part

I defines global parameters. In this sample there is only one definition, which sets up the log file for **sudo** (line 52). This parameter assures the logging of any **sudo** call.

Part II defines some lists of host names. In this sample four host lists are defined: one per operating system (lines 67, 71, and 75) and a fourth one, which includes all other host lists (line 78). The host lists are necessary because the program paths differ on different platforms.

Part III now defines several command aliases. This is the biggest part of the configuration file and it defines several command groups that have to be used by the different roles defined below. The scenario described in this article uses four different roles to control access to WebSphere MQ resources. The first role is called mqshow. Members with this role have only a few permissions on WebSphere MQ objects (lines 96 to 98 and lines 102 to 104). In fact they are allowed only to show authorities, and the WebSphere MQ version (using mqver) and WebSphere MQ error codes (using mqrc).

The next role is named mqoper (see lines 111 to 121 and lines 125 to 135). Members of this role are the operators of the WebSphere MQ queue managers. This means that, in addition to the functions of the role mqshow, they may start and stop queue managers and they are able to manage permissions using **dspmqaut** and **setmqaut**. They are also able to start listeners and trigger monitors, and to save WebSphere MQ objects as well as WebSphere MQ permissions. Members of the role are also able to browse queue contents. Operators are also allowed to run the command line interface, **runmqsc**, to configure and manage WebSphere MQ objects.

The third role I defined is named mqadm (see lines 142 to 155 and lines 159 to 172). I assumed that, in addition to the role mqoper, members of this role should be enabled to create and delete WebSphere MQ queue managers and to dump the WebSphere MQ active logs. This group is also able to get and put messages (it depends on your personal requirements whether or not you create two different roles for administration and operation).

A sort of 'master' role is the predefined group mqm. Members of this group already have nearly any privilege they need. It is not possible to restrict these users in any way. Nevertheless there is one difference between members of this group and the system user mqm. Members of the group mqm – except the user mqm itself – are not able to stop WebSphere MQ processes or clean up IPC segments like shared memory or semaphores. Therefore **sudo** allows members of the group mqm to change to the user mqm using **su** (it could also make sense to allow users of the role mqadm to change to the user mqm and therefore to not put users to the group mqm).

Special situation for the program runmqsc

It is not very helpful to configure the usage of the program **runmqsc** using **sudo**. **Runmqsc** itself controls access to WebSphere MQ objects. Since Version 5.3 of WebSphere MQ on distributed systems, it has been possible to create generic profiles (on hosts this feature already existed in earlier versions). Nevertheless it makes sense to let users (at least members of the role mqoper) call **runmqsc** only through **sudo**. The advantage is that these calls are reported. So, if something happens within the WebSphere MQ configuration, it is possible to find out who was running **runmqsc** and who may be responsible for the problem.

File permissions

After a default installation, most of the WebSphere MQ tools are executable by the whole world. When **sudo** is used to enable the execution of these programs, it is not necessary to have privileges on these programs for users other than mqm. These programs are owned by user and group mqm. Any 'world' permissions have to be removed. Only the program **runmqsc** should be executable by the world, because this program has its own permission management. Use generic profiles, using **setmqaut**, to configure the permissions on the WebSphere MQ objects.

The configuration file, *sudoers*, must be owned by user root and have the group id 0. It also has to be readable by user and group and must not have any further permissions.

How to execute sudo commands

It is quite easy to execute the WebSphere MQ commands using **sudo**. You need to know only where **sudo** resides or have the file location in your search path. Then you have to call the **sudo** program with one of the permitted commands as a parameter. The command has to be entered as shown by **sudo -l**. **Sudo -l** displays all permitted commands. See the output below for the user dummy as a member of the group mqshow:

```
$ sudo -l

User dummy may run the following commands on this host:
(mqm) NOPASSWD: /opt/mqm/bin/dsp*, /opt/mqm/bin/mqrc, /opt/mqm/bin/mqver
```

The user dummy may now execute one of the allowed programs as shown below:

```
$ sudo -u mqm /opt/mqm/bin/mqver

Name:         WebSphere MQ
Version:      530
CMVC level:   p000-L020617
BuildType:    IKAP - (Production)
```

LISTING OF THE SAMPLE SUDOERS FILE

```
########################################################################
#
#    Sample configuration for sudo to configure roles for
#    WebSphere MQ administration. This sample defines four
#    different roles with varying permissions. Each role
#    defines its own Unix group. The roles used
#    here are:
#
#       mqshow: This role is allowed only to look
#                at the WebSphere MQ permissions and may
#                see the WebSphere MQ version and the error
#                codes. Members of this role cannot change
#                anything or even browse a queue.
#
#        mqoper: This role is designed to perform 'normal'
#                WebSphere MQ operation tasks. Members of
#                this role may start and stop a queue
#                manager or a trigger monitor and listener.
#                They cannot create or remove a whole queue
#                manager. Operators are allowed to run the
#                'browse' sample program, but not the 'put'
#                or 'get' samples.
#
#        mqadm:  This role is designed to perform any task
#                that is needed during the administration
#                of WebSphere MQ systems. Some privileges
#                of the role 'mqm', which are not needed for
#                administration but only for solving
#                WebSphere MQ problems, are missing. Members
#                of the role 'mqadm' are able to run the
#                'put' or 'get' sample programs.
#
#        mqm:    This group is predefined in WebSphere MQ.
#                Members of this role have all permissions
#                in WebSphere MQ.
#
#     Author: Hubert Kleinmanns
#     Date:   April 7th, 2004
#
######################################################################


######################################################################
#
#     Part I: Definition of the default parameters.
#
######################################################################

######################################################################
# Definition of the sudo log file.

Defaults logfile = /var/adm/sudolog


######################################################################
#
#     Part II: Definition of the host aliases.
#
```

```
#############################################################

#############################################################
# Definition of your Unix server.
#############################################################

# AIX systems (has to contain a comma separated list of
# your AIX host names or a dummy value)
Host_Alias      HOSTS_AIX=my_aix.my.domain

# Linux systems (has to contain a comma separated list of
# your Sun Solaris host names or a dummy value)
Host_Alias      HOSTS_SUN=my_1st_sun.my.domain,my_2nd_sun.my.domain

# Sun systems (has to contain a comma separated list of
# your Linux host names or a dummy value)
Host_Alias      HOSTS_LINUX=unknown

# All Unix systems (sum of all systems above)
Host_Alias      HOSTS_SERVER=HOSTS_AIX,HOSTS_LINUX,HOSTS_SUN


#############################################################
#
#    Part III: Definition of the command aliases.
#
#############################################################

#############################################################
# Definition of the command aliases for WebSphere MQ.
#############################################################

#############################################################
# Command aliases for members of the sudo role 'mqshow'.

# Definition of all WebSphere MQ display commands on
# AIX systems.
Cmnd_Alias      MQM_SHOW_AIX=      /usr/mqm/bin/dsp*, \
                                   /usr/mqm/bin/mqrc, \
                                   /usr/mqm/bin/mqver

# Definition of all WebSphere MQ display commands on
# Sun Solaris or Linux systems.
Cmnd_Alias      MQM_SHOW=      /opt/mqm/bin/dsp*, \
                               /opt/mqm/bin/mqrc, \
                               /opt/mqm/bin/mqver

#############################################################
# Command aliases for members of the sudo role 'mqoper'.

# Definition of all WebSphere MQ operation commands on
# AIX systems.
Cmnd_Alias      MQM_OPER_AIX=      /home/mqm/bin/saveqmgr, \
                                   /usr/mqm/bin/amqoamd -s, \
                                   /usr/mqm/bin/dsp*, \
                                   /usr/mqm/bin/end*, \
                                   /usr/mqm/bin/runmqlsr, \
                                   /usr/mqm/bin/runmqtrm, \
                                   /usr/mqm/bin/set*, \
                                   /usr/mqm/bin/str*, \
                                   /usr/mqm/bin/mqrc, \
                                   /usr/mqm/bin/mqver, \
                                   /usr/mqm/samp/bin/amqsbcg

# Definition of all WebSphere MQ operation commands on
# Sun Solaris or Linux systems.
Cmnd_Alias      MQM_OPER=      /home/mqm/bin/saveqmgr, \
                               /opt/mqm/bin/amqoamd -s, \
                               /opt/mqm/bin/dsp*, \
                               /opt/mqm/bin/end*, \
                               /opt/mqm/bin/runmqlsr, \
                               /opt/mqm/bin/runmqtrm, \
                               /opt/mqm/bin/set*, \
                               /opt/mqm/bin/str*, \
                               /opt/mqm/bin/mqrc, \
                               /opt/mqm/bin/mqver, \
                               /opt/mqm/samp/bin/amqsbcg

#############################################################
# Command aliases for members of the sudo role 'mqadm'.

# List of all WebSphere MQ administration commands on
# AIX systems.
Cmnd_Alias      MQM_ADMIN_AIX=      /home/mqm/bin/saveqmgr, \
                                    /usr/mqm/bin/amqoamd -s, \
                                    /usr/mqm/bin/crt*, \
                                    /usr/mqm/bin/dlt*, \
                                    /usr/mqm/bin/dmp*, \
                                    /usr/mqm/bin/dsp*, \
                                    /usr/mqm/bin/end*, \
                                    /usr/mqm/bin/runmqlsr, \
                                    /usr/mqm/bin/runmqtrm, \
                                    /usr/mqm/bin/set*, \
                                    /usr/mqm/bin/str*, \
                                    /usr/mqm/bin/mqrc, \
                                    /usr/mqm/bin/mqver, \
                                    /usr/mqm/samp/bin/*

```

```
# List of all WebSphere MQ administration commands on
# Sun Solaris or Linux systems.
Cmnd_Alias      MQM_ADMIN=      /home/mqm/bin/saveqmgr, \
                                /opt/mqm/bin/amqoamd -s, \
                                /opt/mqm/bin/crt*, \
                                /opt/mqm/bin/dlt*, \
                                /opt/mqm/bin/dmp*, \
                                /opt/mqm/bin/dsp*, \
                                /opt/mqm/bin/end*, \
                                /opt/mqm/bin/runmqlsr, \
                                /opt/mqm/bin/runmqtrm, \
                                /opt/mqm/bin/set*, \
                                /opt/mqm/bin/str*, \
                                /opt/mqm/bin/mqrc, \
                                /opt/mqm/bin/mqver, \
                                /opt/mqm/samp/bin/*

#############################################################
# Command aliases for members of the sudo role 'mqm'.

# Set the user id to the "mqm" user.
Cmnd_Alias      MQM_SET_USER=      /usr/bin/su - mqm

# List of all WebSphere MQ commands on AIX systems.
Cmnd_Alias      MQM_ALL_AIX=      /usr/mqm/bin/*, \
                                  /usr/mqm/samp/bin/*

# List of all WebSphere MQ commands on Sun Solaris or
# Linux systems.
Cmnd_Alias      MQM_ALL=      /opt/mqm/bin/*, \
                              /opt/mqm/samp/bin/*


#############################################################
#
#    Part IV: Definition of the sudo roles.
#
#############################################################

#############################################################
# Members of the group "mqm" may change to the user "mqm"
# by entering their own password (not the "mqm" password).
# Other sudo specifications are not necessary, because
# members of the group mqm have any permission they
# need, except stopping processes or removing IPC segments
# directly - which may be necessary, to stop a 'hanging'
# WebSphere MQ queue manager. Therefore members of the group
# mqm are able to become user mqm.

```

```
%mqm            HOSTS_SERVER=(root) PASSWD: MQM_SET_USER
%mqm            HOSTS_AIX=(mqm) NOPASSWD: MQM_ALL_AIX
%mqm            HOSTS_LINUX,HOSTS_SUN=(mqm) NOPASSWD: MQM_ALL

########################################################################
# Members of the group "mqadm" are allowed to execute some
# WebSphere MQ commands without entering a password.

%mqadm            HOSTS_AIX=(mqm) NOPASSWD: MQM_ADMIN_AIX
%mqadm            HOSTS_LINUX,HOSTS_SUN=(mqm) NOPASSWD: MQM_ADMIN

########################################################################
# Members of the groups "mqoper" are allowed to run only
# some WebSphere MQ display, stop, and start commands without
# entering a password.

%mqoper            HOSTS_AIX=(mqm) NOPASSWD: MQM_OPER_AIX
%mqoper            HOSTS_LINUX,HOSTS_SUN=(mqm) NOPASSWD: MQM_OPER

########################################################################
# Members of the groups "mqshow" are allowed to run only
# some WebSphere MQ display commands without entering a
# password.

%mqshow       HOSTS_AIX=(mqm) NOPASSWD: MQM_SHOW_AIX
%mqshow       HOSTS_LINUX,HOSTS_SUN=(mqm) NOPASSWD: MQM_SHOW
```

Hubert Kleinmanns
Senior Consultant
N-Tuition Business Solutions (Germany)

WebSphere MQ Integrator ControlCenter tracing

The simplest (only?) way to trace the WMQI ControlCenter is to call the ControlCenter from the command line. The command to use is:

```
mqsilcc n
```

The *mqsilcc.bat* file is located in the *<mqsi-install>\Tool* directory. The *n* refers to a trace level in the range 0-2, where 0 means no tracing, 1 is for normal tracing, and 2 is for debug tracing.

Once the required trace information has been collected in the log file, *<mqsi-install>\log*, tracing can be stopped. The log file can be formatted by entering:

```
mqsireadlog ControlCenter -t -b ControlCenter -f -o <file>.xml
mqsiformatlog -i <file>.xml
```

Old trace files can be deleted from the *<mqsi-install>\log* directory*.*

Susan Alnutt
WMQ Consultant (UK)

MQ news

SAS has announced a new version of its Enterprise ETL Server, which includes features that simplify core functions in the extraction, transformation, and load (ETL) process, and shorten the time required to make information available to a company.

The product has a wizard-driven user interface, which simplifies how meta data is captured at the transformation level. It also enables ETL routines in the development cycle to be performed in real-time using message queueing products such as WebSphere MQ, as well as through Web services.

For further information contact:
URL: www.sas.com/technologies/dw/entetlserver.

* * *

IBM has announced Version 8.3 of DB2 Content Manager. The new version has enhancements to its content management platform, and new workflow capabilities that help clients automate processes using graphical tools.

Document routing integrates with workflow capabilities to help streamline business processes. The capture and management of XML documents in a common content repository can be automated.

The product integrates with Version 4.1.1 of DB2 Records Manager. Integrating records management with content management allows users to find and access documents, even if they aren't located in the system's own repository. Content Manager 8.3 includes middleware allowing content management to be built into the workflow systems of different corporate business processes, including WebSphere MQ messaging system, WebSphere Application Server, and Tivoli Storage Manager.

For further information contact your local IBM representative.
URL: www.software.ibm.com/data/cm.

* * *

Based on Candle's technology, IBM has announced the Tivoli OMEGAMON XE solution suite, which includes products that monitor and manage zSeries operating systems and subsystems including WebSphere Application Server (WAS), WebSphere Integration Brokers, and WebSphere MQ. It can also monitor and manage z/OS Unix System Services and Parallel Sysplex, z/VM, Unix System Services, CICS, DB2, IMS, zSeries networks, and storage.

For further information contact:
URL: www.ibm.com/ondemand.

* * *