IBM® WebSphere®

Secure Messaging Scenarios with WebSphere MQ

- Using strong authentication
- Using granular authorization
- Implementing revocation checking

T.Rob Wyatt
Glenn Baddeley
Neil Casey
Long Nguyen
Jørgen H. Pedersen
Morten Sætra

ibm.com/redbooks

Redbooks

International Technical Support Organization

Secure Messaging Scenarios with WebSphere MQ

November 2012

SG24-8069-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (November 2012)

This edition applies to WebSphere MQ Version 7.5.

Contents

Notices . . . xi
Trademarks . . . xii

Preface . . . xiii
The team who wrote this book . . . xiii
Now you can become a published author, too! . . . xv
Comments welcome . . . xv
Stay connected to IBM Redbooks . . . xvi

Chapter 1. Introduction . . . 1
1.1 Why read this book . . . 2
1.2 Currency . . . 3
1.3 Scope . . . 3

Chapter 2. What is security . . . 5
2.1 Defining requirements . . . 6
2.2 Security as a system . . . 7
2.3 The security lifecycle . . . 8
2.3.1 Provisioning access . . . 8
2.3.2 Revoking access . . . 8
2.3.3 Monitoring and accountability . . . 9
2.3.4 Ongoing maintenance . . . 9
2.3.5 Recovery . . . 10
2.4 Summary . . . 11

Chapter 3. Authentication and authorization . . . 13
3.1 Relationship between authentication and authorization . . . 14
3.2 Authentication in WebSphere MQ . . . 14
3.2.1 Connection authentication . . . 14
3.2.2 Message-level authentication . . . 16
3.3 Authorization in WebSphere MQ . . . 16
3.3.1 Connection-level authorization . . . 16
3.3.2 Message-level authorization . . . 19

Chapter 4. Connection-level security . . . 21
4.1 Architecture . . . 22
4.2 Authentication . . . 22
4.2.1 Assertion . . . 22
4.2.2 Origin . . . 23
4.2.3 Certificate . . . 23
4.3 Identity resolution . . . 23
4.4 Binding authentication to authorization . . . 24
4.5 Default CHLAUTH rules . . . 24
4.6 Provisioning access . . . 25
4.7 Upgrade and migration . . . 26
4.8 Access control lists . . . 26
4.9 Authorizing topics . . . 27
4.10 Authorizations that grant administrative access . . . 28
4.10.1 Granting +crt authority . . . 28

4.10.2 Granting +set authority on the queue manager . . . 28
4.10.3 Granting +setid or +setall on queues . . . 29
4.11 Common mistakes . . . 29
4.11.1 Unprotected channels . . . 30
4.11.2 Granting access to principals . . . 30
4.11.3 Administrative users with mqm as a secondary group . . . 31
4.11.4 Unquoted asterisks in setmqaut commands . . . 32
4.11.5 Using generic authorizations . . . 33
4.11.6 Granting access to the nobody group . . . 34

Chapter 5. Message-level security . . . 35
5.1 Architecture . . . 36
5.2 Policies . . . 36
5.3 Use cases . . . 37
5.3.1 Business-to-business (B2B) . . . 37
5.3.2 End-to-end encryption . . . 38
5.3.3 Data aggregation . . . 38
5.3.4 Command and control flows . . . 39

Chapter 6. WebSphere MQ security controls . . . 41
6.1 Overview . . . 42
6.2 Operating system and file system resources . . . 42
6.2.1 File system as the root of trust in the server . . . 42
6.2.2 Restrict file system access . . . 43
6.2.3 Restrict access to mqm home directory and tools . . . 44
6.2.4 Limit access to the mqm user ID . . . 44
6.2.5 mqm group membership . . . 44
6.2.6 Files and directories . . . 45
6.2.7 Fully specified names in mqm cron job scheduler . . . 45
6.2.8 Do not administer WebSphere MQ as root . . . 45
6.2.9 Protection of WebSphere MQ backups . . . 46
6.2.10 Increase the size of error logs . . . 46
6.2.11 Archiving error logs . . . 46
6.2.12 Isolation of staging environments . . . 47
6.2.13 Protect user-provided executables . . . 47
6.3 Queue manager local resources . . . 47
6.3.1 Define a system dead letter queue . . . 47
6.3.2 Considerations for dead-letter queue handler . . . 48
6.3.3 Enable event messages . . . 50
6.3.4 Restrict access to remote clustered queues . . . 51
6.3.5 Do not disable WebSphere MQ authorization checks . . . 51
6.3.6 Generic authorization profile names . . . 52
6.3.7 PROCESS and SERVICE objects should use explicit paths . . . 52
6.3.8 Run the command server only when it is needed . . . 52
6.3.9 Limited use of trigger monitors . . . 53
6.3.10 Minimal authority on SYSTEM objects . . . 54
6.3.11 Object names . . . 55
6.3.12 Realistic attribute values . . . 55
6.4 Channels, transmission queues, and communications . . . 56
6.4.1 Use channel authentication rules . . . 57
6.4.2 Disable all incoming SYSTEM channels . . . 57
6.4.3 Always specify a low-privileged MCAUSER . . . 59
6.4.4 Avoid use of put authority context on channels . . . 59

6.4.5 Do not enable automatic channel definition 60
6.4.6 Avoid using a default transmission queue. 60
6.4.7 Avoid use of SERVER channels 61
6.4.8 Restrict access to transmission queues 62
6.4.9 Increase message retry on channels 62
6.4.10 Use the managed listener 63
6.4.11 Specify local address on outbound channels 64
6.4.12 Usage of port numbers 64
6.4.13 Queue manager to queue manager versus clients 65
6.4.14 Separate channels for application messaging 65
6.5 Queues and other objects 65
6.5.1 Restrict access to system default objects 66
6.5.2 Least access authorization model. 66
6.5.3 Authority to SYSTEM.BASE.TOPIC 67
6.5.4 Considerations for dead letter queue and topics. 67
6.6 Applications using WebSphere MQ 67
6.6.1 Avoid setting message context fields 67
6.6.2 Avoid alternate user ID 68
6.6.3 User ID and password fields on client connections. 69
6.6.4 Use segregated input queues 69
6.6.5 Careful use of report messages 70
6.7 Recent changes 70
6.7.1 Dedicated cluster transmission queues 70
6.7.2 WebSphere Message Broker default configuration wizard 71
6.7.3 MCA interception for clients 71
6.7.4 RFC 5280 certificate validation policy 71
6.7.5 FIPS compliance on SSL/TLS and AMS. 71
6.7.6 New CipherSpecs and CipherSuites. 71
6.7.7 NSA Suite B support 71
6.7.8 Distinguished Encoding Rules in SSLPEER and SSLCERTI 72
6.8 Procedural considerations. 72
6.8.1 Software currency 72
6.8.2 Periodic revalidation of security roles 72
6.8.3 Resource monitoring to detect and record security incidents 73

Chapter 7. Operating system specifics 75
7.1 IBM z/OS 76
7.1.1 WebSphere MQ security management 76
7.1.2 TLS/SSL certificate and key repository management 76
7.1.3 Queue sharing groups 76
7.1.4 Channel types have additional values of PUTAUT 77
7.1.5 Separating put and get authority 77
7.1.6 Publish/subscribe security. 77
7.1.7 Certificate sharing in a queue sharing group 78
7.1.8 RESLEVEL security 78
7.2 IBM i 78
7.2.1 Special users 79
7.2.2 Command authorization 79
7.2.3 Key repository 79
7.3 Microsoft Windows 79
7.3.1 Specific profiles for principals and groups in OAM 79
7.3.2 Deleting user IDs or groups 80
7.3.3 Service user ID using active directory. 80

7.3.4 Application event log . . . 81
7.3.5 Securing shared data for multiple instance queue managers on Windows . . . 81

Chapter 8. Scenario preparation . . . 83
8.1 Overview . . . 84
8.2 Servers and network topology . . . 84
8.3 Operating systems and infrastructure software . . . 85
8.3.1 Virtualization . . . 86
8.3.2 UNIX servers . . . 86
8.3.3 Windows servers . . . 87
8.4 Operating system configuration . . . 87
8.4.1 Virtualization . . . 87
8.5 WebSphere MQ installation and configuration . . . 88
8.6 Other software installation and configuration . . . 88
8.7 Naming standards and conventions . . . 88
8.7.1 Host names . . . 88
8.7.2 User ID and group names . . . 89
8.7.3 Queue manager names . . . 89
8.7.4 Channel names . . . 89
8.7.5 WebSphere MQ object names . . . 89
8.8 Certificate authorities . . . 89
8.9 OCSP responder . . . 90
8.10 LDAP server to host CRLs . . . 90
8.11 WebSphere MQ (CMS) keystores . . . 92
8.12 Other certificate tools . . . 94

Chapter 9. Scenario: WebSphere MQ administration . . . 97
9.1 Scenario overview . . . 98
9.1.1 Scenario design . . . 99
9.1.2 Prerequisites . . . 100
9.1.3 Using the additional material for scripts and common variables . . . 101
9.2 Implementing the scenario . . . 102
9.2.1 Preparing the operating system user IDs and groups . . . 102
9.2.2 Creating the queue manager and listener . . . 103
9.2.3 Authorizing queue manager and system objects to enable remote WebSphere MQ Explorer . . . 104
9.2.4 Defining application objects and limited administration authority . . . 105
9.2.5 Providing authority to display all objects . . . 107
9.2.6 Defining a channel for anonymous remote WebSphere MQ Explorer . . . 108
9.2.7 Defining a secure channel for remote administration roles . . . 109
9.2.8 Creating a key repository for queue manager . . . 112
9.2.9 Generating the queue manager certificate and adding it to the key repository . . . 114
9.2.10 Creating a key repository for users . . . 115
9.2.11 Generating the user certificates and adding them to the key repository . . . 116
9.2.12 Building the Java keystore files for users of WebSphere MQ Explorer . . . 119
9.2.13 Setting up the WebSphere MQ Explorer workstation . . . 121
9.3 Configuring WebSphere MQ Explorer for the anonymous administration role . . . 123
9.3.1 Configuring a new queue manager . . . 123
9.3.2 Verifying the display of objects . . . 125
9.3.3 Verifying that the user has no authority to alter objects . . . 125
9.3.4 Removing the queue manager from WebSphere MQ Explorer . . . 126
9.4 Configuring WebSphere MQ Explorer for a limited administration role . . . 126
9.4.1 Configuring the new queue manager . . . 127

9.4.2 Displaying the channel status 130
9.4.3 Verifying that the user can display objects 131
9.4.4 Verifying that the user has authority to alter APP1 objects 132
9.4.5 Verifying that the user has no authority to alter APP2 objects 132
9.5 Configuring WebSphere MQ Explorer for a full administration role 134
9.5.1 Displaying the channel status 137
9.6 Summary 140

Chapter 10. Scenario: Securing IBM WebSphere MQ connections to connect a business partner 141
10.1 Scenario overview 142
10.1.1 Scenario design 142
10.1.2 Scenario flow 144
10.2 WebSphere MQ and WebSphere MQ Internet pass-thru features and practices . . . 145
10.2.1 Application considerations 146
10.2.2 Application queue manager 148
10.2.3 Gateway queue manager 149
10.2.4 WebSphere MQ Internet pass-thru server 149
10.3 Implementing the scenario 150
10.3.1 Application queue manager configuration 150
10.3.2 Gateway queue manager configuration 155
10.3.3 WebSphere MQ Internet pass-thru configuration 162
10.3.4 Application configuration 177
10.3.5 Firewall configuration 178
10.4 Results of testing 179
10.5 Advanced Encryption Standard support in WebSphere MQ Internet pass-thru 183
10.6 Summary 184

Chapter 11. Scenario: Fine-grained cluster security 185
11.1 Scenario overview 186
11.1.1 Scenario design 186
11.1.2 Preparing for the scenario 188
11.1.3 Creating the cluster 190
11.2 Authorizing access using the authority context of user IDs 194
11.2.1 Creating authorization profiles for the full-repository queue managers 195
11.2.2 Creating authorization profiles for the cluster member queue managers 197
11.2.3 Setting the MCAUSER user to block unauthorized access 199
11.2.4 Validating the authorization settings 200
11.2.5 Summary 201
11.3 Authorizing access using queue manager name mapping 202
11.3.1 Creating CHLAUTH records for the full-repository queue managers 202
11.3.2 Configuring the cluster member queue managers 204
11.3.3 Validating the authorization settings 205
11.3.4 Summary 207
11.4 Using SSL for mutual authentication 208
11.4.1 Creating certificates for the cluster queue managers 208
11.4.2 Configuring the cluster queue managers to use SSL 209
11.4.3 Validating the settings 211
11.4.4 Summary 211
11.5 Authorizing access with X.509 DN mapping 212
11.5.1 Configuring X.509 DN to MCAUSER mapping 212
11.5.2 Validating the settings 214
11.5.3 Summary 215
11.6 Authorizing access with X.509 and IP address mapping 215

11.6.1 Validating the settings . . . 216
11.6.2 Summary . . . 217
11.7 Considerations for large clusters . . . 217
11.8 Summary . . . 218

Chapter 12. Scenario: CRL/OCSP certificate revocation . . . 219
12.1 Scenario overview . . . 220
12.1.1 CRL design . . . 220
12.1.2 OCSP design . . . 221
12.1.3 Prerequisites . . . 225
12.1.4 Certificate authorities that are used in this scenario . . . 225
12.2 Certificate revocation . . . 226
12.3 Using certificate revocation lists . . . 227
12.3.1 Configuring CRL in WebSphere MQ . . . 227
12.3.2 Turning off CRL checking . . . 227
12.3.3 For more information . . . 228
12.4 Using Online Certificate Status Protocol (OCSP) . . . 228
12.4.1 Configuring OCSP in WebSphere MQ . . . 228
12.4.2 Making OCSP checking optional . . . 231
12.4.3 For more information . . . 232
12.5 Troubleshooting . . . 232
12.5.1 Security troubleshooting . . . 232
12.5.2 Event messages . . . 235
12.5.3 SSL troubleshooting . . . 238
12.6 Summary . . . 239

Chapter 13. Scenario: End-to-end security using WebSphere MQ AMS . . . 241
13.1 Scenario overview . . . 242
13.1.1 Scenario design . . . 242
13.1.2 Prerequisites . . . 243
13.2 Configuring for first use . . . 244
13.2.1 Queue manager configuration . . . 244
13.2.2 Authorizations . . . 244
13.2.3 Application configuration . . . 244
13.3 Exchanging signed messages . . . 245
13.3.1 Integrity policy definition . . . 245
13.3.2 Test sending and receiving signed messages . . . 246
13.4 Exchanging encrypted messages . . . 248
13.4.1 Key exchange . . . 248
13.4.2 Encryption policy definition . . . 249
13.4.3 Testing the send and receive of encrypted messages . . . 250
13.5 Summary . . . 251
13.6 Further considerations . . . 251

Chapter 14. Scenario: WebSphere MQ AMS revocation checking . . . 253
14.1 Scenario overview . . . 254
14.1.1 Scenario design . . . 254
14.1.2 Prerequisites . . . 256
14.2 Implementing the scenario . . . 256
14.2.1 Queue manager configuration . . . 256
14.2.2 Client configuration . . . 256
14.2.3 Applications . . . 257
14.3 Testing the certificate revocation . . . 257
14.3.1 Test 1: Revoking the receiver's certificate . . . 257

14.3.2 Test 2: Revoking the senders certificate . . . 259
14.3.3 Test 3: Both certificates are renewed . . . 260
14.4 Summary . . . 262
14.5 Further considerations . . . 262

Appendix A. Working with the itsoME message exit . . . 263
A.1 Description . . . 264
A.2 Inbound traffic . . . 264
A.2.1 Message exit functions . . . 265
A.2.2 Configuring the queues . . . 266
A.3 Outbound traffic . . . 266
A.3.1 Configuring the queues . . . 268
A.4 Configuring the exit . . . 268
A.5 Compiling the exit . . . 269
A.5.1 z/OS . . . 269
A.5.2 Linux for System z . . . 272
A.5.3 Linux 64 . . . 273
A.5.4 Linux 32 . . . 273
A.5.5 AIX . . . 273
A.5.6 Solaris SPARC . . . 273
A.5.7 Microsoft Windows . . . 273
A.6 Installing the exit . . . 273
A.6.1 Installation on z/OS . . . 274
A.6.2 Installation on distributed platforms . . . 274
A.7 Design considerations . . . 274
A.8 Debugging . . . 275
A.9 Message exit source . . . 275

Appendix B. Additional tooling for WebSphere MQ Internet pass-thru . . . 291
WebSphere MQ Internet pass-thru start/stop script . . . 292
WebSphere MQ Internet pass-thru control script . . . 293
WebSphere MQ Internet pass-thru security exit (itsoBlockExit) . . . 297

Appendix C. Certificate administration techniques and special WebSphere MQ security checks . . . 307
Managing certificates on z/OS . . . 308
Generating a certificate and key . . . 308
Creating a certificate request . . . 309
Installing a certificate . . . 309
Installing CA certificates in the truststore . . . 309
Creating a keyring . . . 310
Connecting certificates to a keyring . . . 310
Deleting a certificate . . . 311
Listing a certificate . . . 311
Listing a keyring . . . 313
Changing a certificate label . . . 314
Simple z/OS MQ security check . . . 314
MQCHECK REXX script . . . 320

Appendix D. Additional material . . . 341
Locating the Web material . . . 341
Using the Web material . . . 341
Downloading and extracting the Web material . . . 342

Related publications 343
IBM Redbooks 343
Online resources 343
Help from IBM 344

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AIX®
CICS®
DataPower®
DB2®
developerWorks®
FFST™
First Failure Support Technology™
Global Technology Services®
i5/OS™
IBM®
IMS™
MQSeries®
OMEGAMON®
RACF®
Redbooks®
Redbooks (logo) ®
System z®
Tivoli®
WebSphere®
z/OS®

The following terms are trademarks of other companies:

Intel, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, or service names may be trademarks or service marks of others.

Preface

The differences between well-designed security and poorly designed security are not always readily apparent. Poorly designed systems give the appearance of being secure but can over-authorize users or allow access to non-users in subtle ways. The problem is that poorly designed security gives a false sense of confidence. In some ways, it is better to knowingly have no security than to have inadequate security believing it to be stronger than it actually is. But how do you tell the difference? Although it is not rocket science, designing and implementing strong security requires strong foundational skills, some examples to build on, and the capacity to devise new solutions in response to novel challenges. This IBM® Redbooks® publication addresses itself to the first two of these requirements. This book is intended primarily for security specialists and IBM WebSphere® MQ administrators that are responsible for securing WebSphere MQ networks but other stakeholders should find the information useful as well.

Chapters 1 through 6 provide a foundational background for WebSphere MQ security. These chapters take a holistic approach positioning WebSphere MQ in the context of a larger system of security controls including those of adjacent platforms' technologies as well as human processes. This approach seeks to eliminate the simplistic model of security as an island, replacing it instead with the model of security as an interconnected and living system. The intended audience for these chapters includes all stakeholders in the messaging system from architects and designers to developers and operations.

Chapters 7 and 8 provide technical background to assist in preparing and configuring the scenarios and chapters 9 through 14 are the scenarios themselves. These chapters provide fully realized example configurations. One of the requirements for any scenario to be included was that it must first be successfully implemented in the team's lab environment. In addition, the advice provided is the cumulative result of years of participation in the online community by the authors and reflect real-world practices adapted for the latest security features in WebSphere MQ V7.1 and WebSphere MQ V7.5. Although these chapters are written with WebSphere MQ administrators in mind, developers, project leaders, operations staff, and architects are all stakeholders who will find the configurations and topologies described here useful.

The third requirement mentioned in the opening paragraph was the capacity to devise new solutions in response to novel challenges. The only constant in the security field is that the technology is always changing. Although this book provides some configurations in a checklist format, these should be considered a snapshot at a point in time. It will be up to you as the security designer and implementor to stay current with security news for the products you work with and integrate fixes, patches, or new solutions as the state of the art evolves.

The team who wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

T.Rob Wyatt spent five years in IBM Software Services for WebSphere performing consulting engagements for clients in North America and Europe before bringing his security expertise to the WebSphere Connectivity and Integration product management team. T.Rob has worked with WebSphere MQ since 1996 and holds all of the IBM MQSeries® and WebSphere MQ certifications. He is a frequent speaker at conferences and seminars and the

author of the IBM developerWorks® *Mission: Messaging* column. He is based out of Charlotte, North Carolina.

Glenn Baddeley is a Senior Specialist in WebSphere MQ and has been with IBM Global Technology Services® Australia for 15 years. He led the support of WebSphere MQ on several large outsourcing contracts in the Asia/Pacific region and is currently engaged in the delivery of middleware projects and solutions to more than 10 IBM clients. Glenn performs architecture design and consulting and strategic planning. He also sets standards and writes specialized documentation. He does product deployment and tools programming and advises on complex problem solving for many critical business applications that use WebSphere MQ on a wide variety of platforms. This work has given him a deep understanding of the product and a great appreciation of its practical use. Glenn contributes as a Subject Matter Expert on IBM developerWorks and other public MQ forums. Prior to 1997, he worked for a large telecommunications corporation for 14 years as a systems programmer, designing, developing, and supporting customized middleware solutions and ISV Operating System security extensions. Glenn has a Bachelor's degree in Computer Science from Deakin University, Australia. He is the author of IBM SupportPacs MA0K and MA0Z and presented a session on WebSphere MQ Client Security at the IBM Interaction 2000 conference in Australia.

Neil Casey is a WebSphere MQ Subject Matter Expert with IBM in Australia. He has 30 years of experience in IT with more than 15 years of experience with WebSphere MQ on IBM z/OS®, *nix, and Windows. He holds a Bachelor of Applied Science degree with Distinction in Computer Science from the Royal Melbourne Institute of Technology (RMIT). His areas of expertise include WebSphere MQ and other messaging servers, IBM DataPower® XML appliance administration, IBM CICS®, IBM IMS™, and IBM DB2® systems programming, z/OS systems programming, and UNIX administration. He has previously published an article about sorting data in CICS, and has contributed to WebSphere MQ related exit development efforts.

Long Nguyen is a Senior Managing Consultant with IBM Software Services for WebSphere, an IBM organization that assists clients with implementing software integration solutions. His specialized areas of interest include high availability and scalability implementations with WebSphere MQ and WebSphere Message Broker.

Jørgen H. Pedersen is a WebSphere MQ Subject Matter Expert and Senior IT Specialist in Nordic Processor, a subsidiary of IBM. He has more than 30 years of experience with IT and more than 15 years with WebSphere MQ. He holds a degree in Electronic Engineering from Royal Danish Air Force and a Bachelor of Commerce degree. His areas of expertise include system programming for z/OS, *nix, and Windows. His WebSphere MQ expertise covers: planning, installation, configuration, performance analysis and optimization, teaching, mentoring, developer support, troubleshooting and testing on z/OS, IBM i5/OS™, *nix, Solaris, HP NSK and Windows. He has developed some IBM SupportPacs to increase WebSphere MQ security and written extensively on WebSphere MQ Security. He is the inventor of BlockIP2.

Morten Sætra is a Senior IT Specialist in Norway. He has more than 31 years of experience in IBM, and more than 15 years experience with supporting WebSphere MQ clients. His areas of expertise include installing, configuring, tuning, high availability, and teaching WebSphere MQ. He has written extensively on CICS/ESA and MQSeries Integrator V1.

Thanks to the following people for their contributions to this project:

The ITSO project leader: Carla Sadtler
International Technical Support Organization, Raleigh Center

Shawn Tooley
International Technical Support Organization, Raleigh Center

Robert Haimowitz
International Technical Support Organization, Poughkeepsie Center

Andrew Akehurst
Paul Clarke
Morag Hughson
Jacek Krzeminski
Bill Oppenheimer
Jonathan Rumsey
IBM UK

Jacek Krzeminski
IBM Poland

Now you can become a published author, too!

Here's an opportunity to spotlight your skills, grow your career, and become a published author—all at the same time! Join an ITSO residency project and help write a book in your area of expertise, while honing your experience using leading-edge technologies. Your efforts will help to increase product acceptance and customer satisfaction, as you expand your network of technical contacts and relationships. Residencies run from two to six weeks in length, and you can participate either in person or as a remote resident working from your home base.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

- Use the online **Contact us** review Redbooks form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbooks@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYTD Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- Find us on Facebook:

  http://www.facebook.com/IBMRedbooks

- Follow us on Twitter:

  http://twitter.com/ibmredbooks

- Look for us on LinkedIn:

  http://www.linkedin.com/groups?home=&gid=2130806

- Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:

  https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm

- Stay current on recent Redbooks publications with RSS Feeds:

  http://www.redbooks.ibm.com/rss.html

1

Introduction

Before diving headlong into technical details, it is useful to understand a bit more about the objectives and approach taken in this book.

This chapter includes the following topics:

- Why read this book
- Currency
- Scope

1.1 Why read this book

WebSphere MQ V7.1, released in November 2011, included more security features in a single release than any previous version in the history of the product. WebSphere MQ V7.5 improved further by integrating WebSphere MQ Advanced Message Security (AMS) into the base product. These new capabilities are very compelling for users who require highly secure messaging networks. This book is intended to showcase some of those capabilities by using real-world scenarios applicable to almost any environment.

The IBM X-Force 2011 Trend and Risk Report, released in March of 2012, reported a 30% decline in the availability of exploit code, a decrease in unpatched security vulnerabilities, a 50% reduction in cross-site scripting (XSS) vulnerabilities, and a decline in spam. Although this is good news, X-Force reports that attackers displaced from their traditional areas of attack began developing new exploits against previously untapped IT systems and emerging technologies such as mobile. In the face of the evolving threat, companies of all sizes are reassessing their IT infrastructure in a new light.

The year 2008 saw the first large-scale breach of credit card data while in transit, raising the bar for network security. As the industry worked to upgrade communication links, weaknesses in the Secure Sockets Layer (SSL) protocol were discovered, prompting moves to Transport Layer Security (TLS), the successor to SSL. Advances in cryptographic technology continued to render weaker algorithms obsolete. Regulatory bodies responded by deprecating the weakest algorithms and standardizing on newer versions. Certificate authorities worldwide updated their root and signer certificates to use longer key lengths. This in turn sparked a new round of industry-wide network security remediation, which continues today.

WebSphere MQ as of V7.0.1 added support for many new cryptographic algorithms, including the SHA-2 family and elliptic curve cryptography. Many authentication features that were formerly only available in channel exits were added as simple channel configuration in WebSphere MQ V7.1. The same release included many other security enhancements, including authorization of non-local objects, authorization based on remote queue manager names, and role-based access control making it easier to limit administrative access on application and user channels. Network security in WebSphere MQ is better than ever.

But the focus on data in transit does not mean that the threat against data at rest has gone away. Many of the recent breaches pilfered data from memory and file systems. WebSphere MQ Advanced Message Security (AMS) addresses the need for end-to-end security by encrypting messages while in custody of the sending application and decrypting them while in custody of the receiving application. Because the application owners hold the keys, the data cannot leak at any point in between. This approach is superior to solutions such as encrypted disk that provides protection for messages in queue files but leaves messages in plaintext while in the queue manager's memory.

If your enterprise is concerned about messaging security, this book will show you how to incorporate the new features of WebSphere MQ into common messaging scenarios, such as clusters and B2B interfaces, including two scenarios with WebSphere MQ AMS. Because base administrative hardening is the foundation on which all other WebSphere MQ security rests, the book includes a scenario demonstrating that and using the new features. Following the scenarios in the book should give you practical experience that you need to determine how the new security enhancements in WebSphere MQ fit into your overall security plan for data in transit and data at rest.

Although the book was written with WebSphere MQ administrators and developers in mind, other readers should find it useful as well:

- Application architects and designers will find the scenarios useful as a reference to messaging components, security controls, and how these map together in the various topologies depicted.
- Project managers may find the scenarios useful for estimating the scope of work for projects where messaging security is a critical requirement and to inform cost/benefit analyses.
- Enterprise security staff who are unfamiliar with WebSphere MQ should find the book helpful to understand the capabilities of the product and how it fits into the larger enterprise security ecosystem.
- Auditors, assessors, and compliance managers will certainly find the scenarios valuable as references to what types of security controls to look for and where they map out in the messaging network topology.

1.2 Currency

One of the themes of this book is that security is a journey and not a destination. Attacks always continue to evolve. Security that does not also evolve will gradually become ineffective as the attacks catch up to it. This book is based on WebSphere MQ V7.5 and the best information available to the authors at the time of its writing. However, in time attackers may find ways to exploit some of the configurations presented here. At the same time, WebSphere MQ continues to evolve and administrators should review the documentation of major releases to identify any new security enhancements. Readers are encouraged to stay up-to-date with developments in the security field and refer to the IBM WebSphere MQ Security Bulletin for information about recent security vulnerabilities and fixes.

The WebSphere MQ Security Bulletin can be found at `http://ibm.co/WMQSecurityBulletin`.

The WebSphere MQ Recommended Fixes page is `http://ibm.co/WMQRecommendedFixes`.

1.3 Scope

The WebSphere MQ Information Center contains exhaustive documentation of the many security controls in WebSphere MQ and WebSphere MQ Advanced Message Security. In addition, there are many developerWorks articles and conference presentations on the topic of WebSphere MQ security available for download. Given the abundance of information already available, the criteria that determined the scope of this book was the question “what is unique about an IBM Redbooks project that would be difficult to duplicate in a developerWorks article, a conference presentation, or the WebSphere MQ Information Center?”

The answer lies in the structure of an IBM Redbooks residency:

- A team of subject matter experts (SMEs) gathered in one location and dedicated for the duration of the residency
- Availability of enterprise-grade hardware and software that would normally be unavailable to an independent author
- A format that lends itself to deep coverage of individual topics
- An outside-in approach rather than the Information Center’s product-centric approach

With these factors in mind, this book is divided into two main sections. The first section discusses the broader topic of messaging security and how it relates to WebSphere MQ in particular. The perspective is that of designing a secure system in which WebSphere MQ is a component. By looking at security as a system with interdependencies among WebSphere MQ, adjacent technology components, and human processes that drive all of these, it is possible to achieve a much higher effective level of security than if any of these are considered in isolation.

The second section is comprised of several scenarios. Some of these scenarios, for example, the sections on administrative hardening and revocation, are foundation scenarios that subsequent sections will incorporate by reference. Other scenarios are specific business use cases that are commonly encountered and which require security. The choice of scenarios was influenced by feedback from many customers at conferences and services engagements, as well as statistics from Problem Management Reports (PMRs).

The initial list of candidates included scenarios that are commonly implemented and for which PMRs or user feedback indicated a need for better or updated documentation. The list was then prioritized based on which of these scenarios would best take advantage of access to dedicated hardware and software available to the team and that would be difficult to replicate elsewhere. That narrowed the list down to WebSphere MQ B2B, certificate revocation, end-to-end security, and advanced clustering as the primary scenarios, plus a few foundation scenarios, such as administrative hardening, on which the others depend.

The omission of some scenarios that are in high demand is regrettable but intentional. Rather than broad and necessarily shallow coverage of many topics, the approach is deep coverage of relatively few topics. For example, a managed file transfer security scenario would be a welcome addition but it is one that can be designed and tested in a virtual environment and does not require the resources of the ITSO lab. Similarly, a WebSphere MQ Telemetry scenario was considered but preferred practices have yet to emerge for that product. This book was designed so that these and other scenarios can be added as future updates either as part of a formal residency or through the contribution of individual authors.

Another conscious decision was to not duplicate the Information Center. In order to focus on developing, testing, and documenting the scenarios, just enough configuration reference was included in this book to illustrate the scenarios. Readers who are unfamiliar with WebSphere MQ security or who require comprehensive configuration reference will find the Information Center an invaluable companion to this book. Where appropriate, direct links to Information Center topics are provided.

2

What is security

Too often, requirements documents fail to specify what is meant by *security*. The term is often interpreted to mean intrusion protection, but that is just one of many aspects of security.

This chapter examines security broadly as it applies to systems and the position WebSphere MQ occupies in this context.

This chapter includes the following topics:

- Defining requirements
- Security as a system
- The security lifecycle
- Summary

2.1 Defining requirements

Since you are reading this book, it is very likely that you have been reading a requirements document with a line item or two specifying that the WebSphere MQ components must be secure. If that is the extent of the description, you have a normal requirements document, but unfortunately one that is not of much help.

The WebSphere MQ administrator is often left to translate these broad requirements into specific implementation details and the most common approach is to configure fine-grained authorization, occasionally combined with authentication. To obtain sign-off, the administrator demonstrates that authorized users have access while unauthorized users do not. In most cases, the system is then considered secure and the requirements are signed off as complete.

But has it really been secured? It is highly likely that the requirements document was not specific enough:

- Secure against what populations of attackers?
- Is there any capability to detect an attack in real time?
- When an attack is detected, where and how are alarms raised?
- What are the response capabilities if attacked and who is responsible to execute them?
- Does the system provide logging with which to perform forensic analysis after the fact?
- How well does the system support recovery from a security breach?

Before choosing security controls for a given system, the administrator must first ask *what* and *who* the system is to be secured against. In the example, the administrator chose to focus on preventing intrusion by unauthorized users but did not consider other types of attackers. Similarly, the administrator's focus on intrusion prevention mitigates attacks aimed at obtaining access to data or transactions but does nothing to address other categories of attack, such as denial-of-service.

This is not to suggest that every security design must always address all types of attackers or all categories of attacks. Security design must balance cost against risk and some risks do not justify the cost of mitigation. However, any decision to exclude a given risk from consideration should be deliberate and based on an accurate assessment of the cost of the control, the likelihood of an adverse event, and the potential impact of such an event.

At a minimum, then, the administrator should consider the categories of *prevention*, *detection*, *response, and recovery* for each threat considered:

- How is this threat prevented?
- How is it detected, either during the attack or afterward?
- Who are the responders? What are their roles and tasks?
- What are the recovery and business continuity requirements?

These answers are then assessed against the risk, cost, and potential impact if the threat is exploited to arrive at a business case to accept the risk or to implement controls to mitigate it.

Taking the time to think through the larger contexts leads to security models that are broadly effective and have a greater likelihood of actually protecting the enterprise at a cost appropriate to the business risk.

2.2 Security as a system

WebSphere MQ does not operate in isolation. It is middleware, so by definition it occupies a space between two or more communicating applications. If one of these applications is breached, it can be used to attack other applications communicating with the queue manager, or the queue manager itself. Thus, the security of the queue manager is in part dependent on the security of the applications communicating with it.

These same considerations apply at a lower level when considering systems adjacent to the queue manager, such as the host on which the queue manager resides. An attacker who gains privileged access to the host server generally also gains administrative access to WebSphere MQ. Effective security in WebSphere MQ, therefore, depends in part on the security of adjacent systems.

In some cases, security depends on components that are not directly adjacent to the queue manager and can be easily overlooked. For example, when using certificates signed by a certificate authority (CA), security depends on whether the CA imposes strict controls over certificate generation and is highly resistant to attacks. Similarly, if the file system under the queue manager is backed up, security of WebSphere MQ becomes dependent on the ability of the backup system to keep configuration settings confidential through encryption and tight control of the encryption keys. Effective security in WebSphere MQ, therefore, depends in part on the security of non-adjacent infrastructure systems.

Consider also that the best security can be thwarted by human fallibility. Users often choose passwords that are easy to remember, but because they are easy to remember they are easy for someone else to guess or to crack. Users sometimes circumvent physical and procedural security measures simply for convenience, such as when they share passwords or attach them to their monitors with sticky notes. Frequently, WebSphere MQ administrators over-authorize legitimate users. Adding all the application service accounts and human users to the Administrative group is a common example of this overauthorization. Effective security in WebSphere MQ, therefore, depends in part on users and administrators adhering to basic security hygiene principles.

Security is also constantly evolving; the attacks are always improving and the defenses must keep pace. The attacker need only succeed once, and has the luxury of a relatively static target. The system administrator must get it right every time and faces a constantly evolving threat. In the last few years alone, certificates from at least three different certificate authorities (that we know about) were successfully forged. These breaches were caused by human failures, but even the underlying cryptography is under attack. The MD5 hash has been broken and all versions of Secure Sockets Layer (SSL) are now obsolete. Transport Layer Security (TLS), the successor to SSL, has been updated three times since its release. Effective security in WebSphere MQ, therefore, depends in part on the ability of the administrator to apply patches and upgrades in response to evolving technology and threats.

If effective security of WebSphere MQ depends on the security of the applications, the adjacent systems, the security of non-adjacent infrastructure, the basic security hygiene of human users, and the ability of administrators to evolve in step with technology and attackers, WebSphere MQ security is only properly described in the context of this larger system. Although approaching security design in this systemic context is no guarantee of success, approaching security as an isolated, one-time task does guarantee failure.

The best possible case when security is implemented in isolation as a one-time task, assuming all the configurations are perfect, is that attack technology will evolve and eventually render today's security controls obsolete. Or, as is frequently observed in the field, poor security in one of the interdependent systems or human factors renders the best security configurations moot.

By taking a systemic approach to security, you are more likely to identify all the important touchpoints for security hardening and properly validate the assumptions upon which the WebSphere MQ security depends. Rather than focusing exclusively on WebSphere MQ configuration, this perspective encourages you to consider external factors, such as file system permissions, the robustness of a certificate authority, and a myriad of others which comprise the total system.

This systemic approach also means that security is an ongoing process and not just an implementation. There is a maintenance aspect that requires you, in your role as the security administrator, to keep up-to-date with developments in the industry and in the products in which you specialize. In the event that a cryptographic protocol is broken, a vendor publishes a security patch, or some other security-relevant event occurs, you should be aware of these changes and factor them into the maintenance plan for WebSphere MQ.

2.3 The security lifecycle

Because security is usually treated as a secondary consideration, many architects and planners approach it from a functional perspective with a plan that includes a line item to lock down the system and a second line item to provision access. The full security lifecycle is rarely considered at this stage, so secondary functionality, such as access revocation or patch application, tends to fall through the cracks. This section discusses some aspects of the security lifecycle that are best considered during the initial planning stages yet often neglected.

2.3.1 Provisioning access

Because a secure system allows little or no access to anonymous users, it is obvious that a process to provision new access is required. What is less obvious is the requirement to validate that access is granted only to authorized users. An example of this is when provisioning user access to message queues. The usual sequence of events is that a new user or application attempts to connect to a queue manager and is refused. The administrator then runs the authorization commands and the user is able to connect. Usually, success is declared at this point and the access request is marked complete.

What is often overlooked is whether the change enabled other users, including those users without authorization, to also connect. This example is in a category of failure called *overauthorization*. Cases of overauthorization are difficult to detect because they fail silently. Legitimate users have the access that they need and so do not complain. Unauthorized users rarely realize that they have been incorrectly authorized, and even when they do, they do not usually alert the administrator. Of course, the attacker who discovers the overauthorization never complains. If there is no formal scan or test for overauthorization, the first indication of a problem might be a breach.

When planning the controls and processes for provisioning access, ensure that tests and scans for overauthorization are included.

2.3.2 Revoking access

During the implementation of a new system, the requirement to provision access is obvious. Often the requirement to revoke access is overlooked until the first time that an authorized user changes roles or leaves the organization. That point is a bit late in the process to consider access revocation for the first time. Do not wait until you need it to discover that you do not have it.

Access revocation is a primary security use case. So, treat it as such and design the revocation functionality alongside the access provisioning design. And be sure to test that it works. In addition to deletion of accounts, other examples of revocation include certificates, physical access, shared passwords[1], and door codes.

2.3.3 Monitoring and accountability

If there were an adverse security event on the network, how would you know? WebSphere MQ emits event messages to special event queues and writes to two separate error logs. These messages and error logs are very useful in detecting an attack in progress or for forensic analysis, but only if they are configured properly and monitored. Similarly, WebSphere MQ can emit event messages when configuration changes are made. These event messages capture the change that was made and the identity of the user who initiated the change, thus providing a measure of accountability.

In the default configuration, WebSphere MQ is tuned for a balance of reliability and performance. Event messages are disabled and the error log file size is set to a value that minimizes resource usage. However, a busy queue manager or determined attacker can cause the important information to *age off* (disappear from the end of the error logs) in a matter of seconds. Some approaches to address this issue are increasing the size of the error logs or monitoring and archiving them as they age off.

Similarly, most types of WebSphere MQ event messages are disabled by default. From a security perspective, enabling authorization events and configuration events can be very useful. Once they are enabled, a program can monitor the event queues and log the messages, perhaps taking appropriate action in real time. IBM Tivoli® OMEGAMON® XE for Messaging is an example of such a program. Several IBM Business Partners sell monitoring agents and it is also possible to write a custom monitoring agent by using the APIs and documented features of WebSphere MQ. Enabling event messages without an agent to monitor them results in the event queues filling after which no further event messages are produced for the full queues. Active monitoring of event queues is, therefore, essential when events are enabled.

When designing security for WebSphere MQ, consider the need for monitoring and accountability. Most systems will benefit from increasing the size of the error logs. For new users or small networks, consider using SupportPacs to monitor and log the event messages. Where the potential risk or impact is great, or for larger networks, consider a commercial monitoring agent.

2.3.4 Ongoing maintenance

Another aspect of security that is not always obvious at design time is the need to apply updates and patches in a timely manner. Consider the key renegotiation bug that was discovered in the SSL protocol in 2009. Vendors quickly updated their software to address the issue and published patches. If your software, from any vendor, uses SSL and pre-dates the patch, it may be vulnerable.

In the case of WebSphere MQ, IBM published the fix for this problem in fix packs for all then-current versions. No patch was published for out-of-date versions. If you are running an out-of-date version or a supported version that pre-dates the patch and you use SSL channels or HTTPS listeners, you may have a latent vulnerability.

[1] This comment is not an endorsement of shared passwords, merely an acknowledgement of their use.

Of course, this is just an example to illustrate the point that keeping a system secure may involve updating or patching it from time to time. As part of system security design, ask the question "*How long would it take and how many resources would be required to update the WebSphere MQ software to apply a high priority security patch?*" Consider not just the queue managers but also the adjacent systems such as client applications, monitoring, instrumentation, and desktop tools. Consider how you would prioritize such an effort. Are the queue managers ranked by service level or some other indication of criticality? Do the teams that are involved have sufficient capacity to perform an unplanned patch, or would an urgent task displace other work? What is the potential impact of displacing that other work?

Whether it takes a week, a month, or a year is not as important as whether that length of time and resource allocation is appropriate to the business risk involved. If an honest assessment concludes that it would take six months to roll out a patch, the determination of whether that is good or bad depends on many factors such as the potential impact of the risk, whether there are other mitigating controls to limit the exposure, the nature of the patch, and the severity of the problem.

In some regulatory environments, this time frame is decided for you. For example, the Payment Card Industry Data Security Standard (PCI-DSS) requires critical patches to be applied within a month of their release by the vendor. A savvy auditor will want to verify not just that you are compliant today but that you have a plan in place to apply patches within the time window that is required to remain compliant.

An effective security design will consider the issue of how urgent patches and updates will be applied. You may require the ability to patch in 30 days as per the PCI-DSS, or at the other extreme, you may decide to deal with the problem on an ad hoc basis when and if it arises. As explained in other sections of this book, the right answer has more to do with making a conscious, deliberate decision than with the actual target goal that you select.

2.3.5 Recovery

The design of a system should consider what happens when components fail and the manner in which they fail. It is helpful here to illustrate the concepts by using a purely mechanical example: the lowly ball bearing.

Traditional metal ball bearings have limited heat tolerance. As they wear, they become rough and pitted. This condition generates more heat, which can make them so soft that they deform. Over time, they may wear away, break apart, or melt. As the process proceeds, performance gradually degrades until eventually the system fails altogether.

Compare this situation to ceramic ball bearings, which have outstanding tolerance to heat and can run at much higher speeds than metal ball bearings. However, they wear differently. Ceramic ball bearings do not deform, melt, or wear away over time. When they fail, they simply disintegrate. The performance curve for these ceramic ball bearings looks very different. Everything is fine until the moment of failure and then everything stops.

The metal ball bearing is inexpensive and has limited heat tolerance but it degrades gracefully. The ceramic ball bearing is expensive and has high heat tolerance but degrades catastrophically. Neither of these scenarios is inherently bad or good, but the maintenance and recovery approaches differ greatly.

The same is true of security. Effective security designs assume that failure will occur and plan accordingly. Prefer solutions that degrade gracefully over those solutions that degrade catastrophically. Where a security control can degrade catastrophically, the design should include regular inspection and maintenance to prevent such failures. Unlike a ceramic ball bearing, security that fails catastrophically can do so silently. Frequent updates and inspections can avert problems.

Although the business drivers for high availability and disaster recovery solutions usually cite natural disasters, human error (such as cutting a communications trunk line), or acts of war, security breaches are equally valid use cases. It may seem odd to find a discussion of high availability and disaster recovery in a book that is focused on security, but in the event of a breach, these high availability and disaster recovery systems are the systems on which you are most likely to restore service.

With that in mind, one criterion for security design would be the degree to which compromise of one node compromises adjacent nodes. For example, if the primary server is compromised, will the attacker need to work as hard to break into the secondary server? If the servers can administer each other, the compromise of one server means that both servers are compromised. Thus, if security considerations are factored into the high availability and disaster recovery designs, the approaches that are selected are more likely to provide administrative isolation.

It is also important to write a formal plan for security incidents. Know in advance what different roles and tasks are, who fills those roles, and how to coordinate among them. Practice this formal plan once or twice a year, just as you practice disaster recovery. When dealing with business partners, make sure to maintain current contact information and make sure that the incident procedures cover security incidents and not just disaster recovery failover.

2.4 Summary

In summary, keep these factors in mind when designing security:

- Assume that a breach will occur and design accordingly.
- Include multiple security layers to slow or prevent breaches from spreading.
- Design security to degrade gracefully to a secure state rather than degrading catastrophically to an open state.
- Inspect and test for overauthorization.
- Test negative cases as well as positive cases.
- Integrate security into the high availability and disaster recovery plans.
- Create a written incident plan and test it regularly.

3

Authentication and authorization

Security controls are often classified according to the type of service that they provide, such as privacy, integrity, accountability, authentication, or authorization. Of these controls, authentication and authorization are the foundation on which everything else is built, so it is worth examining these two controls in greater detail.

This chapter contains the following topics:

- Relationship between authentication and authorization
- Authentication in WebSphere MQ
- Authorization in WebSphere MQ

3.1 Relationship between authentication and authorization

In most use cases, authentication and authorization are both required elements of a security design. *Authentication*, or the verification of a claimed identity, is of little use unless controls are applied that differentiate one user's access from another's. Likewise, *authorization*, or the enforcement of access controls, is of little use without validated identities against which to apply rules.

There are exceptions, however. For example, it is common to provide limited, inquire-only access to anonymous WebSphere MQ users. This use case is accomplished by defining authorization rules to enforce the limited access and then applying those rules to anonymous connections. The opposite case applies to authenticated administrators. It is critical to strongly authenticate administrators, but in WebSphere MQ, it is not possible to restrict such users with authorization rules. So, anonymous users are a case of authorization without authentication and administrative users are a case of authentication without authorization.

All other use cases require both authentication and authorization. These use cases include any case where multiple applications or groups of users require varying permissions to queues and topics. Each entity requesting access is first authenticated to ensure that the identity presented is genuine. Next, access control is applied so that the requestor gets only the appropriate access and nothing more. This approach supports everything up to the most complex, fine-grained, multi-role security models, although, in practice, simpler designs are better.

3.2 Authentication in WebSphere MQ

WebSphere MQ provides authentication for connections and, when WebSphere MQ Advanced Message Security is used, on the messages themselves.

3.2.1 Connection authentication

The original design of WebSphere MQ was as a transport stack that was used by local programs. Since all connections to the queue manager originated locally, the identities in the operating system's process table were the basis for authorization. WebSphere MQ did not perform any authentication since any process requesting a connection must have been authenticated by the operating system. Queue manager connections that arrive over shared memory rather than the network are known as *bindings mode* connections. Current versions of WebSphere MQ continue to support bindings mode connections and these connections rely on the local operating system's process ID for authentication and identity.

The WebSphere MQ authentication of remote connections works differently because the local process ID that is associated with the connection belongs to the process running the channel rather than the remote entity. This situation is true whether the connections are from other queue managers, client applications, or interactive users connecting with desktop client software, such as WebSphere MQ Explorer. Although the remote user of the client software has been authenticated at the remote host and the connection may originate from a trusted workstation, the queue manager cannot distinguish that connection from an unauthorized connection unless some authentication takes place at the queue manager. It is therefore necessary for the administrator to arrange for the queue manager to authenticate inbound connection requests.

One method for authenticating connection requests is to use a channel security exit. A *channel exit* is a point during the processing of the channel's code where an external program is called. There are several types of channel exits that perform different functions. Security exits are called during the startup of a channel. Security exits provide the administrator with an opportunity to perform arbitrary authentication checks outside the scope of what WebSphere MQ natively provides. Security exits are called on the sending and receiving sides of the channel and can negotiate complex exchanges, but it is the exit on the receiving queue manager's side that is responsible for accepting or denying the connection.

WebSphere MQ V7.1 incorporated three of the most common security exit functions into the queue manager, thus eliminating the need for security exits in many cases. The new functions are implemented as CHLAUTH rules and can filter inbound connection requests based on one or more of three criteria:

- Asserted identity (queue manager name or user ID) that is presented by the channel
- IP address of the remote partner that is requesting the connection
- X.509 distinguished name of the certificate that is presented by a Secure Sockets Layer (SSL)/Transport Layer Security (TLS) channel

Although the first two of these criteria do not qualify on their own as true authentication, they are useful in combination with other security controls. For example, the identity that is presented by a client application might be considered trustworthy if it originates from a certain IP address, belonging to a server in a locked datacenter, and is one of an enumerated list of acceptable possible user IDs. Individually, the physical controls on the datacenter, the IP address filtering, and the user ID filtering would not be sufficient authenticators but, in some cases, the combination of all of these authenticators could be considered appropriate.

Where stronger authentication is required, X.509 certificates are the preferred solution. For example, the preferred authentication for remote administrative connections is to use X.509 certificates with mutually authenticated channels. WebSphere MQ channels contain an SSLPEER attribute that filters connection requests based on fields in the distinguished name in the certificate. The SSLPEER matching string can be fully qualified or may contain wild cards. The combination of certificate validation performed in the SSL or TLS protocol and the SSLPEER matching performed by the queue manager resolves a connection down to a single certificate or one of a set of acceptable certificates.

Since WebSphere MQ authorizations are enforced against user IDs defined to the local operating system, an IP address or X.509 distinguished name are not entities that can be directly authorized. So, one of the functions of WebSphere MQ authentication is to resolve the authentication credentials that are provided by the requestor to a user ID that the queue manager can use for authorization checks. The CHLAUTH rules do this by mapping any of the three authenticators, the remote user ID, the IP address, or the certificate distinguished name, to a local user ID that is defined to the operating system. The resolved identity is then placed in the channel's MCAUSER attribute as the channel starts. It is the MCAUSER that is used for all authorization checks once the CHLAUTH rules and security exits are complete.

Important: The initial connection is made using the process ID of the message channel agent, and an internal call is used to switch the user ID when the first message is received. This is necessary because security flows between the exits could alter the MCAUSER at any point until the first message is passed. As a result, it is possible to see the channel in RUNNING status with an MCAUSER other than that which is expected, even one that is not a valid user ID, such as *NOACCESS.

3.2.2 Message-level authentication

In addition to authentication of connections, WebSphere MQ provides authentication of individual messages. When using WebSphere MQ Advanced Message Security, messages are signed by the sender using a private key. This cryptographically binds the identity of the sender to the message. The consumer of the message is then able to authenticate the message using the sender's public key. The WebSphere MQ administrator can configure policies that control whether messages must be signed and potentially contain a list of authorized senders of signed messages. In addition to signing messages, WebSphere MQ Advanced Message Security can also encrypt messages. Encrypted messages can be authenticated in the same manner as signed messages, based on the sender's public key. As with signed messages, the policy may optionally include a list of authorized senders. The policy must also contain a non-optional list of intended recipients. The reason for this list is that for each recipient, the message encryption key is itself encrypted using the public key of the intended recipient and then added to the message. To perform that step requires foreknowledge of who the recipients are as well as possession of their public keys.

When deciding whether to use message-level security in your network, it is helpful to consider the asynchronous aspect of messaging. One process connects to the queue manager to produce a message, and a second, completely independent process connects to the queue manager to consume the message. These two processes are loosely coupled and have no direct knowledge of one another. They may not even connect at the same time or to the same queue manager. The only identity information that the consumer application can access is the identity information that is contained in the message itself. If the message is changed in transit or produced by an unauthorized sender, the consumer has no way to detect this situation with a plaintext message. In this case, the consuming application trusts the identity fields in the message but cannot verify their authenticity.

However, if the message is encrypted or signed, the consuming application is assured that the message was not changed in transit and that it originated from an authorized sender. Whereas connection security can be thought of as forming a perimeter around the messaging network, this ability to bind the sender's identity to the message forms a perimeter around individual messages.

3.3 Authorization in WebSphere MQ

The previous topic discusses authentication - the methods by which WebSphere MQ validates identities that are associated with remote connections. This topic is concerned with authorization - the functions in WebSphere MQ that enforce access policies against queue managers and their internal resources, such as queues and topics. As with authentication, WebSphere MQ provides authorization functions at the connection and message layers.

3.3.1 Connection-level authorization

The queue manager authorizes API calls based on the identity that is associated with the connection so it is important to understand which components can connect to a queue manager, as well as the means by which their identities are resolved and associated with the connection.

Types of connections

Applications can connect directly to a queue manager by using shared memory (what is known as a *bindings mode* connection), or over the network by using the WebSphere MQ client. With the bindings mode connection, the identity that is associated with the connection is the operating system identity of the connected process. With a client connection, the associated identity is resolved during the connection negotiation. The identity that is associated with the connection may be the one presented by the application, or the administrator can choose between hardcoding the user ID in the channel definition or mapping it from the authentication fields. (Recall from the previous topic that the authentication fields are the asserted identity, the IP address, and the X.509 distinguished name.)

In addition to client channels, a queue manager has several types of channels that facilitate communication between queue managers. By default, these channels run with administrative authority but that setting can be overridden to restrict operation of the channel. As with client channels, the administrator has the options of hardcoding the ID in the channel definition or mapping it from the authentication fields.

For all these types of authentication and for all channel types, the resolved identity is placed in the channel's MCAUSER attribute. It is this value that is used for authorization checks.

Associating identities with connections

In all of these cases, a channel where the MCAUSER attribute is not mapped, but instead left blank during the connection negotiation, allows administrative access. Therefore, the administrator wanting to secure the queue manager must set, map, or validate the MCAUSER value of all channels that are not intended for the exclusive use of administrators.

This step meaningfully ties the authentication to the authorization. Without this step of populating the channel's MCAUSER, it is possible to unintentionally grant administrative privileges to a non-administrative user. To illustrate, consider a user connecting over an SSL channel and authenticating with an X.509 certificate. Although the certificate is held by a non-administrative user, there is no connection between that certificate and the local user ID that is associated with that user. Because the WebSphere MQ API allows that user to assert an identity during the connection, the administrator must provide the mapping between the X.509 distinguished name and the local user ID. This mapping is accomplished by using a CHLAUTH mapping rule that derives the user ID based on the certificate distinguished name and places that value in the channel's MCAUSER attribute. This mapping of the channel's MCAUSER attribute prevents the user from asserting an arbitrary user ID during the connection request in order to obtain administrative access.

Additional connection authorization controls

As of WebSphere MQ V7.1, the default settings for a new queue manager disable remote access for all SYSTEM channels and all client channels with three default CHLAUTH rules as shown in Example 3-1. When upgrading an existing V7.0 or earlier queue manager to V7.1 or higher, these rules will not be defined for you.

Example 3-1 Default CHLAUTH rules for new V7.1 and higher queue managers

```
SET CHLAUTH(*) TYPE(BLOCKUSER) +
       USERLIST(*MQADMIN) +
       WARN(NO)

SET CHLAUTH(SYSTEM.*) TYPE(ADDRESSMAP) +
       ADDRESS(*) +
       MCAUSER( ) +
```

```
       USERSRC(NOACCESS) +
       WARN(NO)

SET CHLAUTH(SYSTEM.ADMIN.SVRCONN) TYPE(ADDRESSMAP) +
       ADDRESS(*) +
       MCAUSER( ) +
       USERSRC(CHANNEL) +
       WARN(NO)
```

The first rule in Example 3-1 on page 17 blocks all administrative access on all client channels. The generic ID *MQADMIN represents all administrative users on the local platform. For example, on UNIX and Linux servers, it represents the mqm ID and all members of the mqm group. On Microsoft Windows platforms, it represents any users in the mqm and Administrators groups. Because of this rule, it is necessary to explicitly provision remote access for administrators by using client-based tools, such as WebSphere MQ Explorer.

The second rule in Example 3-1 on page 17 blocks all access to all channels whose names begin with `SYSTEM.` Although the SYSTEM.DEF.* and SYSTEM.AUTO.* channels exist to serve as templates from which new channel definitions inherit, the inbound instances of these channels are fully functional. Many people use these channels out of convenience and both IBM and third-party documentation frequently suggest using SYSTEM.DEF.SVRCONN for client channels. The reason is because writers of articles and documentation know that this channel will be defined and using it in documentation eliminates the need to walk you through defining a new channel. But from a security perspective, using pre-defined SYSTEM channels is a bad practice. The fact that these names are well known is the very reason to lock them down. This CHLAUTH rule preserves the template function of the SYSTEM.* channels without the exposure of making them functional.

The third rule allows remote connections to the SYSTEM.ADMIN.SVRCONN channel and maps the MCAUSER channel attribute based on the user ID asserted by the remote client. Users of WebSphere MQ Explorer typically connect to this channel and because of the mapping, the channel will run with the authority of the connected user. Note that this rule is one of two ADDRESSMAP rules that match the same channel. Where multiple rules match, the most specific match wins. This rule is more specific than the SYSTEM.* ADDRESSMAP rule so it takes precedence for requests to start the SYSTEM.ADMIN.SVRCONN channel.

The *MQADMIN rule discussed earlier is a BLOCKUSER rule and it applies independently of the two ADDRESSMAP rules. So, even though the ADDRESSMAP rule allows users to connect to SYSTEM.ADMIN.SVRCONN and assert any user ID, if the ID they assert is administrative, the connection will be refused. Note that this does not prevent users from spoofing one another when connecting to SYSTEM.ADMIN.SVRCONN, so long as they do not present an administrative ID.

If stronger authentication of users is required, the administrator should configure TLS on the channels, use a security exit, or do both. Since the SYSTEM.ADMIN.SVRCONN channel is not defined by default, the administrator will need to define a channel if remote access using WebSphere MQ Explorer or other client-based tools is required.

API authorization

Once the channel is created, and the connection request has passed all of the applicable CHLAUTH rules, the MCAUSER that was resolved during the connection negotiation is used for all subsequent API authorization checks. Although the checks are performed against a user ID, the access control lists are associated with groups. WebSphere MQ uses the local operating system's identity repository. When WebSphere MQ needs to resolve an ID or a group, it performs an operating system call.

WebSphere MQ does not check passwords. It simply asks the operating system "Does this ID exist?" and then, if the answer is yes, it asks "What groups is the ID enrolled in?" Once the group enrollment is determined, the queue manager checks the access control lists for all applicable groups against the object the call was executed against and the options requested.

3.3.2 Message-level authorization

In addition to per-connection authorization that is provided by WebSphere MQ, authorization on a per-message basis is provided by WebSphere MQ Advanced Message Security. *It is important to understand that WebSphere MQ Advanced Message Security enhances the security in the base product rather than replacing it*. Connection security grants access to queues whereas the message-level security provides the ability to apply policy to individual messages as they are put to or consumed from queues.

Message policies

There are three possible message policies:

- No protection
- Integrity - messages must be signed
- Privacy - messages must be encrypted and signed

Integrity and privacy policies can contain a list of authorized senders. When the message is consumed, it is automatically requeued to the error queue if it did not originate from one of the authorized senders.

In addition to signing, a policy may specify encryption of messages. Because messages are encrypted using the public keys of the intended recipients, a policy that specifies encryption must contain a list of the authorized recipients. Because only the intended recipients hold keys that can decrypt the message, any attempt by an unauthorized recipient to consume the message returns random bytes.

Use cases

The function of message-level authorization is better understood in the context of its use cases. Although there are many, the following three use cases are common and arise from very different requirements:

- Identity aggregation: Some network topologies deliver messages from multiple sources over the same channel. The connection-level security on the channel must be configured to allow the channel to deliver messages to the set of queues that is the union of legitimate destinations of all authorized senders. It is difficult in this case to prevent Sender A from putting messages onto Sender B's queues. WebSphere MQ AMS can enforce policies that the receiving application will only consume messages from authorized senders.

- End-to-end encryption: In some high-security environments, it is necessary to prevent even the WebSphere MQ administrators from viewing message payloads. To meet this requirement, the sending and receiving applications each hold the necessary encryption keys. Because the WebSphere MQ administrators do not hold a valid key and are not listed as authorized recipients, there is no possibility of the message being rendered in plaintext while in the custody of the queue manager.
- Command and control flows: It is a good idea, and in some regulatory regimes a mandatory requirement, to protect command and control flows of sensitive applications. WebSphere MQ AMS can insure that the application only acts on commands from legitimate administrators and can optionally encrypt the exchanges as well.

Of the three use cases, the first use case mitigates a security risk due to topology, the second use case addresses the classic data protection requirement, and the third use case addresses the need to both encrypt and authenticate high-value administrative communications. What they all have in common is a requirement to enforce access controls to a degree that is not possible by using only connection security.

Authorization scope

One of the advantages of WebSphere MQ Advanced Message Security is the ability to enforce authorizations against WebSphere MQ administrators. In V7.1 and earlier versions of WebSphere MQ, the administrator had the ability to put or get messages from any queue. If a protection policy specifies encryption and the administrator is not one of the named senders or recipients, it is possible to remove messages from the queue but not to read them. Any message that the administrator puts onto that queue will be rejected before the application ever sees the message.

As of WebSphere MQ V7.5, the Advanced Message Security functionality has been extended further so that even an administrator will receive an error when attempting to access a queue in a way that would violate the stated protection policy.

4

Connection-level security

This chapter is intended to provide background information about security design considerations for IBM WebSphere MQ channels as of version 7.5. Because security evolves over time, this chapter deliberately tries to avoid excessive duplication of information from the WebSphere MQ Information Center. For detailed explanations of the various security controls that are discussed here and throughout the book, see the Information Center.

This chapter contains the following topics:

- Architecture
- Authentication
- Identity resolution
- Binding authentication to authorization
- Default CHLAUTH rules
- Provisioning access
- Upgrade and migration
- Access control lists
- Authorizing topics
- Authorizations that grant administrative access
- Common mistakes

4.1 Architecture

WebSphere MQ provides two types of channels to accept remote connections. Message Channel Agent (MCA) channels facilitate communications between queue managers. Message Queue API (MQI) channels allow applications to execute API calls over the network. Both types of channels follow the same pattern of authenticating the connection request, resolving an identity for the remote partner, binding the identity to the channel, and performing authorization checks that are based on the bound identity. After the identity is resolved and bound to the channel, it persists for the lifetime of the connection.

Another architectural aspect that impacts security design is the asynchronous nature of messaging. The producer of a message and the consumer of that message each have their own independent connection. Neither the producer nor the consumer has direct access to the context of the other's connection and, in fact, the producer and consumer do not need to be connected simultaneously. The only information that the consumer has about the sender is that which is carried in the message itself. The queue manager fills in message context information using the resolved identity of the connection, but a suitably authorized remote partner can override the queue manager and set the context directly. Alternatively, a message might be intercepted between the producer and the consumer and modified.

The implication is that connections form the secure perimeter of the messaging network on which the integrity of messages depends. Security design must therefore focus on restricting permissions of lightly authenticated connections and strongly authenticating any connections with update or administrative access.

4.2 Authentication

Authentication is usually defined as validating that an identity presented is genuine. Although this meaning suggests a definitive result, in fact there are degrees of authentication. WebSphere MQ natively supports three different levels of authentication: assertion, origin, and certificate. These levels may be used individually or in any combination. Since authentication is normally used in combination with other security controls, it is the entire system of controls that should be considered when evaluating the suitability of one authentication method versus another.

4.2.1 Assertion

The most basic form of authentication in WebSphere MQ is that the requestor asserts its identity in the connection request and the queue manager accepts the identity presented. Although this form is not true authentication, there are many valid use cases. For example, assertion is very useful for connections that are restricted to inquiry commands. Although such a connection could safely be anonymous, the use of assertion allows the differentiation of multiple channel instances by the user name that is associated with each.

For client channels, the identity asserted is the user ID of the remote partner. For MCA channels, the identity asserted is the name of the remote queue manager.

4.2.2 Origin

Origin-based authentication makes a trust decision that is based on where the connection originated. In this case, the origin of the request is the IP address of the requestor, which is why this method is sometimes referred to as *IP filtering*. Although this method is a weak form of authentication, it should be considered in the context of other security controls. For example, if the originating IP address is on the same subnet as the queue manager and both nodes reside in a locked datacenter, the physical security of the datacenter and the routing architecture of the network combine with the IP filtering to provide a level of assurance that the identity presented is genuine.

4.2.3 Certificate

Certificate authentication is the strongest type of authentication that is currently offered by WebSphere MQ. During a connection negotiation, the requestor provides an X.509 certificate and an arbitrary string of data signed by the private key that is associated with that certificate. The queue manager decrypts the data using the public key of the certificate. If the data is successfully decrypted, the queue manager knows with a very high level of assurance that the requestor holds the private key to the certificate presented. The queue manager can then use the fields in the certificate such as Common Name, Organization, and Organizational Unit to map the certificate identity to a local user ID that can be used for authorization.

Although this type is considered strong authentication, what is provided is an assurance that the requestor holds a certificate that is trusted, not an assurance that the requestor's identity is genuine. This difference is a subtle but vital distinction because it highlights the importance of certificate management. The trustworthiness of a certificate depends on the robustness of the certificate authority, the rigor of the certificate management procedures, and the security of the file system where the certificates and private keys are stored.

4.3 Identity resolution

Authorization checks in WebSphere MQ are always based on identities that are known to the queue manager's local operating system. However, the queue manager has no method to determine whether the identity presented by a remote connection partner is resolved from the same identity repository that is used by the local operating system. Although the local system might have an ID called `app1`, the same ID presented from a remote partner may represent a different application or user.

In some cases, the identity that is presented may not be defined in the local operating system at all. For example, when IP filtering is used to authenticate a connection, the IP address has no direct equivalent as an operating system identity. Similarly, when using certificates, the ID represented by the certificate is in the form of an X.509 distinguished name and has no correlation to the simple string ID or Windows Security Identifier that is used by WebSphere MQ.

In all of these cases, the identity of the connection must be resolved to a local operating system identity. WebSphere MQ provides three ways to resolve the identity:

- Hardcoding the channel's MCAUSER attribute
- Mapping the identity using CHLAUTH rules
- Setting the identity using a channel exit

Channels with a single authorized remote connection partner can be configured with the desired identity hardcoded in the channel's MCAUSER attribute. The MCAUSER of channels configured this way cannot be overridden by the remote connection partner.

When there are multiple connection partners for a channel, CHLAUTH rules can be used to dynamically map the resolved identity to the local operating system identity. A textbook example is authentication of WebSphere MQ Explorer using certificates and then mapping the certificate distinguished names to local operating system accounts.

The third identity resolution mechanism is that a channel security exit can set the MCAUSER during the connection negotiation. There is no restriction on how the exit can authenticate the user so long as the identity that it places in the channel's MCAUSER matches a local operating system account.

4.4 Binding authentication to authorization

WebSphere MQ channels run as child processes of the queue manager. This means that the channel *always* has administrative privileges on the queue manager to which it connects. In most cases, however, the remote connection partner must not run with administrative privileges. The mechanism by which the channel enforces restricted access to the queue manager is the MCAUSER attribute of the channel. This value is used for authorization checks.

Setting MCAUSER either when the channel is defined or dynamically during channel start-up is what binds the resolved identity to the authorization checks. It is the administrator's responsibility to insure that channels intended for non-administrators always have a low privileged value set in the MCAUSER and that channels intended for administrators strongly authenticate connections.

The default CHLAUTH rules ensure that this policy is enforced for new queue managers but the security can be compromised in the course of provisioning new access. Since the MCAUSER is where the identity is bound to the channel, it is advisable to always set an MCAUSER value in the channel definition. For channels where an exit or CHLAUTH rule does not override the MCAUSER, the value set must be one under which the channel can and should run. Where the MCAUSER is dynamically set during channel start-up, the MCAUSER in the channel definition should be one that blocks access to the queue manager. This provides an additional control so that if the exit or CHLAUTH rule is disabled, the channel will fail to a secure state rather than wide open.

4.5 Default CHLAUTH rules

When a new queue manager is created at V7.1 or V7.5, it contains three default CHLAUTH rules as shown in Example 4-1. One of these is a BLOCKUSER rule and two are ADDRESSMAP rules. Multiple rules from each type can map a given connection request but a precedence hierarchy determines which rule determines success or failure.

Example 4-1 Default CHLAUTH rules

```
SET CHLAUTH(*) TYPE(BLOCKUSER) USERLIST(*MQADMIN) +
       DESCR('Default rule to disallow privileged users')

SET CHLAUTH(SYSTEM.*) TYPE(ADDRESSMAP) USERSRC(NOACCESS) +
       DESCR('Default rule to disable all SYSTEM channels')
```

```
SET CHLAUTH(SYSTEM.ADMIN.SVRCONN) TYPE(ADDRESSMAP) +
       USERSRC(CHANNEL) ADDRESS(*) +
       DESCR('Default rule to allow MQ Explorer access')
```

The first of these rules disallows all administrative connections on all channels now defined or that may be defined in the future. This rule establishes a baseline policy of *deny all connections for SVRCONN channels*, which can then be overridden with more specific rules that permit access to selected channels. This rule blocks access only on those channels where a user ID can be flowed from the remote partner to the queue manager. It has no effect on channels between queue managers - specifically those channels of type RQSTR, RCVR, and CLUSRCVR are not impacted by this rule.

The second rule disables access to all SYSTEM channels, regardless of type. Channels named SYSTEM.DEF* or SYSTEM.AUTO.* are intended as templates and should not be used for live connections to the queue manager. Despite the existence of many examples that specify its use, if security is a concern, SYSTEM.DEF.SVRCON should be disabled along with all the other SYSTEM channels. This rule insures that the SYSTEM channels remain unusable even if the first rule is deleted or modified. The use of CHLAUTH rules to disable the SYSTEM channels is not meant to replace the more traditional methods, such as setting MAXMSGL(1) and MCAUSER('*noaccess'). The employment of multiple overlapping security controls provides *defense in depth* and ensures that failure of one control does not result in compromise of the entire system.

The third rule opens access to the SYSTEM.ADMIN.SVRCONN channel. The name SYSTEM.ADMIN.SVRCONN matches two of these default ADDRESSMAP rules and this rule, which is the most specific, takes precedence. This ADDRESSMAP rule tells the queue manager that *when a connection request arrives from the specified address, map it according to the USERSRC setting*. In this case, the address is a wildcard that matches all possible connections and the USERSRC(CHANNEL) says to use the ID that is presented by the client. Although this rule allows clients to present any arbitrary ID, the default BLOCKUSER rule that was discussed earlier denies the connection if the ID presented is an administrator. The combination of these rules is that only non-administrative users are eligible to connect to SYSTEM.ADMIN.SVRCONN.

At this point, the SYSTEM.ADMIN.SVRCONN channel will accept non-administrative connections, but on a newly defined queue manager, no users will have been granted access to connect. Even though the channel will accept the connection, the queue manager will return an authorization error and the channel will end. Since this channel is the only channel on the queue manager that will accept connections, these rules mean that a new queue manager assumes a secure posture by default. In order to accept remote connections, the administrator must still authorize the appropriate groups with the ability to connect and execute API calls.

4.6 Provisioning access

Since the new queue manager does not allow any remote access by default, it is necessary for the administrator to explicitly provision any remote access that is required. In the case of non-administrative users, the SYSTEM.ADMIN.SVRCONN channel will accept connections but the users must be defined to the local operating system and access control lists must be created by using **`setmqaut`** or **`SET AUTHREC`** commands. If the users are granted inquire-only privileges, no authentication rules may be needed.

However, if users can perform more substantive operations, such as putting messages, destructive gets, or even browsing if the messages are sensitive, additional authentication is required.

Since WebSphere MQ administrators have an extraordinary amount of access, including remote code execution on the queue manager's host server, remote administrative channels should always use certificate authentication. This guideline does not necessarily exclude the use of additional controls, such as IP filtering, it emphasizes only that the strongest possible authentication is used for administrative channels.

4.7 Upgrade and migration

When upgrading a queue manager from a version that did not support CHLAUTH rules to V7.1 or V7.5, CHLAUTH rule checking on the upgraded queue manager is disabled and the rules are not defined. To enable CHLAUTH rule checking after an upgrade, it is necessary to manually define the default rules and alter the queue manager with the following command:

```
ALTER QMGR CHLAUTH(ENABLED)
```

An alternative is to migrate rather than to upgrade. In that scenario, a new queue manager at the target version is built and all queues, topics, channels, and other objects are defined. Rather than a conversion, a cutover is executed at the designated time. In the event of a problem, the fallback is to resume using the original queue manager.

Because two queue managers should not have the same name, migration requires the affected applications to be free from any dependency on the queue manager name. This approach is the preferred approach for clustered applications and for service-oriented architectures (SOAs) in general, so many applications will find migration to be a viable alternative.

4.8 Access control lists

Throughout this book, the term *access control list* is used to refer generically to the result of either **setmqaut**, **SET AUTHREC** commands, or their equivalent Programmable Command Format (PCF) commands. The **setmqaut** commands are issued directly on a command line, the **SET AUTHREC** commands are issued through the runmqsc interpreter, and the PCF commands are sent as messages to the command queue where they are processed by the command server. All of these types of commands result in authorization profiles that are stored as persistent messages in SYSTEM.AUTH.DATA.QUEUE.

It is important to understand the relationship between CHLAUTH rules and access control lists. The CHLAUTH rules are part of channel start-up negotiation whereas the access control lists are checked only when an API call is executed. There is some slight overlap because the very last step in channel start-up is the connect API call; however, it is useful to think of access control lists as occupying a lower position in the communication stack relative to CHLAUTH rules.

Because access control lists are the last line of defense against unauthorized access, the queue manager assumes a secure-by-default posture. Only administrators have access by default to a newly created queue manager. For all other users and applications, it is necessary to define access control lists before they can connect and interact with queue manager objects. Therefore, one of the first tasks after building a new queue manager, or when on-boarding a new application, is to create authorization control lists.

Access requests are always made by a specific account, but authorization rules are based on groups[1]. When an access request is made, the queue manager uses the operating system to resolve the groups to which the requestor belongs, then looks up access control list entries that are associated with these groups in order to return a permit or deny decision. Therefore, all accounts and groups requiring access to the queue manager must be defined to the operating system on the server where the queue manager is hosted. This requirement can be extrapolated out to a secondary requirement that each security role must have a corresponding operating system group defined.

4.9 Authorizing topics

Authorization checks of most objects in WebSphere MQ are fairly intuitive. The queue manager matches the object name against its access control lists, resolves the most specific match, and returns an immediate decision that is based on that access control list entry. Because topic authorizations are checked differently as compared to other objects, they warrant some additional explanation.

Consider the topic hierarchy in Example 4-2 with a base topic of Price. A publisher authorized to the `Price` topic is implicitly authorized to publish on all subtopics, including `Price/Fruit` and `Price/Vegetables`. But a publisher authorized only to `Price/Fruit` is not authorized to publish to `Price/Vegetables`.

Example 4-2 Topic hierarchy

```
Price/
Price/Fruit/
Price/Fruit/Apples
Price/Fruit/Oranges
Price/Vegetables/
Price/Vegetables/Broccoli
Price/Vegetables/Kale
```

But what happens when multiple levels of the hierarchy contain conflicting authorizations? For example, assume that a subscriber is authorized at `Price/`, then denied access to `Price/Fruit/` by another authorization rule. In the authorization model that is used by queues, the application could access all topics below `Price/` except for those topics at `Price/Fruit/` and lower. But in the authorization model used by topics, the subscriber is granted access to everything under `Price/` including `Price/Fruit/` and all its subtopics.

With queues, the *most* specific authorization rule takes precedence. With topics, the *least* specific authorization rule takes precedence.

When the queue manager checks the topic hierarchy for authorizations, it begins at the point in the hierarchy where the publication or subscription is opened. It then checks for any authorization rule that grants access. If none is found at that level, it checks the next higher level. Checks continue in this fashion until either authorization is granted or the top of the topic tree is reached. If no authorizations are found during any of these checks, access is denied.

To avoid confusion, design the topic hierarchy such that authorizations are granted at a single level, and remember that authorization at any point includes authorization to all points lower in the hierarchy.

[1] The exception is Microsoft Windows where it is possible to authorize per user ID, but this option does not scale so group-based authorization is preferred there, as well.

4.10 Authorizations that grant administrative access

WebSphere MQ authorizations do not affect administrators so any authorizations that you set will be intended for non-administrative applications and users. It is important to understand which authorizations can be granted routinely and which confer partial or full administrative authority.

4.10.1 Granting +crt authority

When a user or application creates an object in WebSphere MQ, they are automatically granted full API access to that object. An example is the creation of a dynamic queue. The user who created that queue has access to all of the API options, including the ability to set message context on messages written to the queue, to clear the queue, or to delete the queue.

This behavior has security implications if the user is granted rights to create queues at will. The risk is that the user can create an alias over the System Command Queue. Because the user will be granted all API options on that alias, they could use their ability to set message context to spoof the administrative user ID on command messages, which would grant them full administrative rights to the queue manager.

There is no need to grant **+crt** authority to allow the use of dynamic queues. Merely granting **+dsp +get** on the appropriate model queue is sufficient.

Also, **+crt** is a system-wide authority. Although the syntax of the **setmqaut** and **SET AUTHREC** commands allows **+crt** to be specified on a fully qualified queue name, the effect is to grant the permission system-wide. The following example is a perfectly valid command:

```
setmqaut -m MYQMGR -g APP1 -t Q -n APP1.REPLY.QUEUE +put +get +inq +crt
```

This command does not grant permission to create APP1.REPLY.QUEUE exclusively as might be inferred from the command syntax. It actually grants the right to create any queue of any name or any type.

The **+crt** command is valid for object types other than queues. Although the escalation to administrative privileges is not as direct as with queues, granting **+crt** for other object types is not recommended. For example, a user who can create a service can cause any arbitrary command to be executed on the server that hosts the queue manager, and these commands will be executed using the queue manager's administrative account. Similarly, a user with **+crt** permissions for queue manager objects could create new queue managers on the host server, which could then be used as a staging area to extend the attack further into the network on the underlying host server.

Do not grant permission to create new WebSphere MQ objects, **+crt**, to non-administrators.

4.10.2 Granting +set authority on the queue manager

Any application connecting to a queue manager should have connect and inquire rights. The ability to set queue manager attributes in combination with the ability to put a message on the command queue allows a user to manage authorizations, which in turn opens a path to escalate to full administrative rights.

The line command for managing authorizations is **setmqaut** and only members of the mqm group (or platform equivalent) have authorization to run it. However, the equivalent programmable command format (PCF) version of the command requires only that the user have **+set** authority on the queue manager.

Do not grant **+set** permissions on the queue manager to non-administrative users.

4.10.3 Granting +setid or +setall on queues

Although this permission does not directly grant administrative rights, it should be used with caution. If a user or application can set the identity context information in a message, programs that depend on that information for authorization decisions can be passed false credentials. One such program that evaluates the identity context of messages is the queue manager's command server. Therefore, it is a well-known practice that **+setid** or **+setall** must not be granted on the command queue to non-administrative users.

What is less well known is that the same permissions on transmission queues may allow messages with a spoofed identity to be placed on the command queue of adjacent queue managers.

Many products and applications use a command server, though they may call it by a different name. WebSphere MQ File Transfer agents and WebSphere Message Broker instances are examples of components that make authorization decisions based on message identity context and listen on command queues. Granting **+setid** or **+setall** on these queues to non-administrators opens the ability to escalate to administrative privileges.

In addition to creating a local exposure, **+setid** and **+setall** may create an exposure on adjacent queue managers. Channels between queue managers have the option to authorize message put operations based on the identity carried in the message header. In the channel definition, the PUTAUT attribute controls this behavior. When PUTAUT(CONTEXT) is set, the channel makes authorization calls on each destination queue based on the identity in the message. If the channel on the target queue manager is set to use context authority and the sending user on the remote queue manager has the ability to set the message identity context, this can escalate to full administrative rights on the adjacent queue manager.

Do not routinely grant **+setid** or **+setall** to non-administrative accounts.

Where these rights are granted, strictly limit them to only those queues where the permission is required. Such grants should be exceptional and therefore subject to an elevated review and approval.

4.11 Common mistakes

This section captures some of the security mistakes that have been observed repeatedly in real-world implementations.

4.11.1 Unprotected channels

It is a surprisingly common error that an administrator will provision a channel for an application or business partner, make the effort to configure SSL peer checking, a low-privileged MCAUSER, and possibly an exit, but will leave channels intended for trusted applications unprotected. The reasoning is that the internal application is trusted because it is on the internal network, is managed within the same enterprise, and resides in a locked datacenter, which is also managed internally. What has been overlooked is that the application or business partner can specify any channel on the queue manager on their connection request. There is nothing to restrict them to their authorized channel unless the administrator configures it.

Another error in this category is that the administrator will protect all the SVRCONN channels but leave RQSTR, RCVR, and CLUSRCVR channels unprotected. The reasoning is that the developer or business partner only needs client access and does not have a queue manager. The problem with this approach is that it is quite easy to download the WebSphere MQ Server trial and create a queue manager. If the attacker can determine the name of your inbound channels and has a TCP route to the queue manager, the attacker can define a channel and attempt to connect. Whether the attacker succeeds depends entirely on the security controls that have been implemented.

Preferred practice: Any channel that is defined on the queue manager, which allows update access, or even browse access if the information is sensitive, requires security controls. All non-administrative channels should have an MCAUSER set or mapped. If the queue manager has both internal-facing and external-facing channels, simple IP address filtering can be used to control which channels an external partner can use.

4.11.2 Granting access to principals

Over the years, many articles, examples, and even the WebSphere MQ Information Center have provided examples of the **setmqaut** command using the **-p** option rather than the **-g** option. The **-p** option, which stands for *principal*, requires the administrator to supply a user ID against which to apply authorizations whereas the **-g** option requires a group. Because the examples have used **-p** and because the syntax supports it on all platforms, many people use it without suspecting that it does not necessarily behave as they expect.

When **setmqaut** is executed against a principal, the primary group of that account is resolved by the queue manager and the authorizations are then applied to that group. For example, assume that the ID `user1` has a primary group of `staff` and is granted rights to connect to a queue manager with the following command:

```
setmqaut -m QMGR -p user1 -t qmgr +connect +inq +dsp
```

Rather than authorizing only `user1`, the command actually authorizes the `staff` group and all users in that group, which is probably not what was intended.

Even if the intent was to authorize the `staff` group, it is better to explicitly use the **-g** option and specify the intended group than to use the **-p** option and rely on the account's group enrollment remaining stable over time. The **setmqaut** commands (and as of WebSphere MQ V7.1, the equivalent **SET AUTHREC** commands) are normally saved to scripts that can be rerun to update the queue manager or restore it. If **-p** has been used and the account's primary group enrollment changes between script runs, the result will be to leave the prior authorizations untouched and to create a new set on the new primary group.

The exception to this approach is the Windows platform where the authorizations are stored against Windows Security Identifiers, also known as SIDs. However, even on this platform, the **-p** option might not behave as expected. The queue manager resolves the specified user ID to an SID using the name resolution functions of Windows Active Directory. The queue manager presents the string representation of the ID and Windows attempts to resolve it first locally, then by querying any domain controllers that it can resolve. In the event that the same ID is defined in multiple domains, it will have a different SID in each domain. So, the ID to which WebSphere MQ assigns the authorizations is actually unique to a specific domain.

A problem can arise when the Windows server resolves the ID to a different domain because the same ID is defined in a domain that is higher up in the search order, or simply because the domain controllers resolve in a different order after a reboot. In these cases, Windows will return a different SID, causing an authorization failure. To the administrator looking at the string representation of the ID, it will appear to have resolved correctly. This type of error is very subtle and can be difficult to diagnose.

To avoid this problem when using the **-p** option on Windows, always use the **user@server** syntax when the ID is defined locally, or the **user@domain** syntax where the ID is resolved to a specific domain. For example, the command above executed on a Windows server might use the following syntax:

```
setmqaut -m QMGR -p user1@devdomain -t qmgr +connect +inq +dsp
```

This command will assure that the ID is always resolved to the expected domain. So long as that domain remains available, the queue manager will authorize the account as intended.

Preferred practice: Always use the **-g** option rather than the **-p** option for **setmqaut** commands on platforms other than Windows.

Preferred practice: On Windows platforms, when using the **setmqaut -p** option, always qualify the account with the domain or server that is expected to resolve it in the format **user@server** or **user@domain**.

4.11.3 Administrative users with mqm as a secondary group

On platforms where an account has primary and secondary groups and where an account is enrolled in the mqm group, if mqm is not the primary group, overauthorization failures usually result. The reason is that when a user creates an object, such as a queue or topic, WebSphere MQ grants that user full authority over that object. On platforms other than Windows where authorizations are based on groups rather than on principals, the queue manager resolves the user's primary group and grants the authorizations to that.

For example, consider an administrator's account that is primary in the `staff` group and secondary in the `mqm` group. WebSphere MQ authorizes the `staff` group with full privileges to every object created by that administrator. Since the group `staff` likely contains every human user that is defined to the system, the result is that every user is fully authorized to every object on the queue manager.

One approach to mitigate this risk is to make sure that `mqm` group enrollment is always primary. Although this approach works, it can be brittle and when it fails, the failure is silent and may not be detected for some time.

A real-life example involved a department whose account management policy stipulated that all user accounts must be primary in the `staff` group. An exception was made for WebSphere MQ administrators to make `mqm` the primary group and `staff` one of the secondary groups. Over time, a well-intentioned account administrator noticed the discrepancy and updated the group enrollments to match the policy such that the administrators' primary group became `staff` and `mqm` became one of the secondary groups. After that event, the WebSphere MQ network became less secure over time as each new object that was defined was authorized to all users. Although the administrative team defined additional authorization rules describing the intended security policy, nobody noticed that the users already had access to the new objects. By the time that the problem was noticed several years later, thousands of objects were authorized to `staff`, rendering the security ineffective.

Another approach to mitigating this risk is to refrain from enrolling administrators in the `mqm` group and instead require them to perform administration using the `mqm` ID. Typically, the administrator signs on under their own ID and then uses the **`sudo`** command to become `mqm`. Although this approach resolves the problems associated with primary/secondary group membership, it requires administration to be performed at the command line or a central tool if accountability to a specific administrator is required because multiple connections from client-based tools would be indistinguishable from one another. Even if desktop tools, such as WebSphere MQ Explorer maintain a log, that log exists on the workstations of several users. A central tool, such as IBM Tivoli OMEGAMON XE for Messaging, addresses the logging issue by maintaining a log of each user's activity.

Preferred practice: If account policy allows enrollments in the `mqm` group, ensure that the `mqm` group is always primary and verify its status at least weekly. Consider adding checks to the administrator profiles so that group membership is verified during each login.

Preferred practice: If account policy requires that administration tasks are performed by using the `mqm` account, use **`sudo`** or another technique to allow the administrators to become mqm. Do not share the `mqm` password and allow direct login to the account.

4.11.4 Unquoted asterisks in setmqaut commands

Although this problem is rare, its symptoms can be subtle and quite difficult to diagnose. When a wildcard character is used on a command line, the operating system shell substitutes the names of matching files in the local directory. The resulting command line depends on whether any files match. Consider the following command:

```
setmqaut -m QMGR1 -g APP1 -t queue -n SYSTEM.* +inq +dsp
```

If the local directory has no files that match the string `SYSTEM.*`, the command line is passed to `setmqaut` unchanged and has the intended effect. A profile is created that matches all queues with the prefix SYSTEM.

If the same command is executed in a directory that contains a single matching file, that file name is substituted on the command line. For example, if the local directory contains a file named `SYSTEM.LOG`, the resulting command line passed to **`setmqaut`** contains that file name:

```
setmqaut -m QMGR1 -g APP1 -t queue -n SYSTEM.LOG +inq +dsp
```

Because the command-line syntax is correct, the command runs. In versions of WebSphere MQ that require a local object definition on which to attach authorization rules, the command will fail with an error code stating that no queue by that name exists. In V7.1 and later, the command succeeds but the authorization profile that is created is not what was intended.

Executing the same command in a directory that contains multiple matching files results in a syntax error. For example, if the local directory contains files named `SYSTEM.LOG` and `SYSTEM.TXT`, the resulting command line that is passed to **`setmqaut`** contains both of these file names:

```
setmqaut -m QMGR1 -g APP1 -t queue -n SYSTEM.LOG SYSTEM.TXT +inq +dsp
```

The syntax error occurs because **`setmqaut`** expects only one string parameter for the **`-n`** option.

Thus the same script can have three different possible results, two of the three being failures, depending on the contents of the local directory. A user may even experience different results over time, for example by redirecting output of the script to a log file with a name that happens to match one of the command line wild-card strings. In some cases, different users can obtain different results where the same script consistently works for one user while it consistently fails for another.

Preferred practice: Quote or escape any wildcard parameters as required by your platform's shell interpreter.

4.11.5 Using generic authorizations

WebSphere MQ has several generic authorization specifiers, each of which represents a set containing multiple explicit rights. For example, the `alladm` authority is expanded by the queue manager to the individual rights `chg`, `clr`, `dlt`, `dsp`, `ctrl`, and `ctrlx`, as of WebSphere MQ V7.5. When the command is run, these individual rights are recorded rather than `alladm`. The other generic authorization specifiers at the time of this publication include `all` and `allmqi`.

Although use of these generic authorization specifiers does provide some convenience, it breaks the security principle of explicit configuration. Reliance on implicit configuration is brittle. It assumes that the implicit conditions will remain stable over time, which is *not* always the case. To create security configurations that are robust, minimize assumptions by using explicit specifiers in cases where a change in the underlying assumption could cause the security control to fail to an insecure state.

An example from WebSphere MQ history is the `allapi` specifier. In earlier versions of WebSphere MQ, granting `+allapi` on the queue manager expanded to `+connect` and `+inq`. As of WebSphere MQ V7.0, the `+set` permission was added to the queue manager objects. Among other privileges, `+set` on the queue manager grants the ability to manage authorizations using PCF commands. Therefore, the `+allapi` specifier, which had been safe for non-administrative users in previous versions, became an administrative privilege and unsafe to grant to non-administrative users for WebSphere MQ V7.0 and higher. However, the specifiers for the individual rights did not change. Scripts that explicitly specified `+connect` and `+inq` continued to provide exactly the intended results.

This section applies to authorizations and not to object profiles. When `+set` was added to the `+allapi` specifier, the security control failed to an insecure state: users were inadvertently granted the right to manage authorizations. However, the use of generic object profiles such as SYSTEM.** is not only beneficial but often necessary. It is common to create a hierarchy of profiles with broad grants of restricted privileges using generic profiles with more permissive rights granted on an enumerated list of more specific profiles. Although it is possible that this approach may fail to an unsecure state with the addition of new objects, the alternative of using only explicit profiles does not scale well. As the list grows larger, the effort of managing it and risk of mistakes exceeds the marginal benefit gained by avoiding generic profiles.

Preferred practice: Avoid using generic authorizations, such as `all, alladm`, or `allapi`.

4.11.6 Granting access to the nobody group

For purposes of authorization, WebSphere MQ treats the `nobody` group as a universal group in which all accounts are enrolled. Therefore, granting permissions to that group grants the permissions to all accounts, regardless of their actual group enrollments. For example, granting +connect on the queue manager to the `nobody` group allows connections from any user.

There are use cases in which it is useful to grant universal but limited access to WebSphere MQ. However, this decision should be deliberate and used with caution. There have been cases in which administrators have granted permissions to the `nobody` group in the mistaken belief that this action would restrict access. As with other overauthorization failures, this failure is not detected unless negative testing is performed.

Preferred practice: Do not routinely grant access to the `nobody` group. In cases where such access is helpful, strictly limit the permissions granted. They should provide the least amount of privilege that is required to accomplish the business task and in no case should they grant a privilege that is administrative or can be escalated to administrative rights.

5

Message-level security

This chapter is intended to provide background information about security design considerations at the message level through signing and encryption. These functions are provided by IBM WebSphere MQ Advanced Message Security (AMS). WebSphere MQ Advanced Message Security was formerly a stand-alone product. It is now incorporated into WebSphere MQ as of V7.5. Because security evolves over time, this chapter deliberately tries to avoid excessive duplication of information from the information center. For detailed explanations of the various security controls discussed here and throughout the book, see the online information center.

This chapter includes the following topics:

- Architecture
- Policies
- Use cases

5.1 Architecture

One of the design goals of WebSphere MQ Advanced Message Security (AMS) was that it should function without impact to applications, thus enabling message protection for homegrown or commercial applications. This goal was achieved by inserting WebSphere MQ AMS between the application's API call and the queue manager executing that call. Outbound messages are intercepted, signed, and optionally encrypted prior to the handoff to WebSphere MQ. Inbound messages are received from WebSphere MQ to the WebSphere MQ AMS layer, decrypted and validated as necessary, and then handed off to the receiving application. This architecture is capable of very sophisticated cryptographic protection of messages and enforcement of fine-grained policies that authorize individual senders and recipients of messages, with few or no changes to the program logic.

The system is based on public/private key encryption with keys stored in standard X.509 certificates, which in turn are stored in a platform-appropriate keystore. These certificates are the same type of certificate used for WebSphere MQ Secure Sockets Layer (SSL) and Transport Layer Security (TLS) encryption, and in fact the application can utilize the same certificates and keystores for both if desired.

When a queue is opened, the WebSphere MQ AMS components request the protection policy for that queue from the queue manager. Messages put to the queue are then signed and possibly encrypted according to policy and using the keys in the application's keystore. Similarly, messages that are retrieved from a queue are processed according to policy and any that do not conform to that policy are rejected and moved to an exception queue.

As of WebSphere MQ V7.5, these capabilities are integrated directly into the WebSphere MQ client and server code. In addition to making WebSphere MQ AMS easier to install and use, the tight integration to the queue manager brings enhanced internal security. For example, in the current version, any update to a policy, or even the queue containing the policies, is mediated by the queue manager and auditable through configuration event messages.

5.2 Policies

Messages that are sent using WebSphere MQ AMS can be in plaintext, signed by the sender, or signed and encrypted. The quality of protection (QOP) that is used is based on policies that are defined by the WebSphere MQ administrator. These policies are associated with queues and each queue can have exactly zero or one policy. Queues with no policy, which is the default, have an effective policy where QOP = `None`.

In addition to specifying the quality of protection, policies can specify the algorithms that are used for signing and encryption operations. When encryption is specified, the policy must also specify all authorized recipients and the sender must have the public key of those recipients in the local keystore. Policy can also specify authorized senders and this specification is enforced for messages that are signed or encrypted.

On a queue manager where message sender and receiver are both attached to the same local queue, a single policy applies to both applications. However, the more common scenario is that the sender and recipient are attached to different queue managers that are, in turn, connected using WebSphere MQ channels. In this case, one policy is defined for the sender's QAlias or QRemote and another policy is defined on the recipient's local queue. The two policies must match in order for messages to flow between the applications.

5.3 Use cases

The need for message encryption arises from the asynchronous messaging model and how it differs from synchronous messaging, such as Secure Hypertext Transport Protocol (HTTPS). When an HTTPS request arrives at the server, the user's credentials are evaluated and the resolved identity and attributes are placed in the environment of the process handling the request. The resolved identity and attributes are passed to the request handling process as a single unit. The context metadata of the request becomes part of the request.

With asynchronous messaging, the model is different because the producer and consumer of a message are separated by both time and distance and do not share each other's connection contexts. The resolved identity of the processes producing and consuming messages are not available to one another beyond whatever information is carried in the message. The queue manager where the message is created places some resolved context information (fields, such as the user ID of the producer and the application name) into the message header, but in some cases this is not sufficient.

For example, the account name or program name of the message producer may be meaningless on the system where the message is consumed. Assuming that the information does resolve locally to a meaningful value, there is no guarantee that a plaintext message is authentic. It may have been intercepted and changed or it may not have originated from the legitimate sender.

Message-level security addresses these problems by cryptographically binding the sender's identity, as represented by the distinguished name in an X.509 certificate, to the message. If desired, the message can also be encrypted for specific recipients who are also identified by X.509 certificates. The ability to bind sender and recipient identities to a message and make it tamper-proof facilitates the creation of policies as to who can send or receive messages, and the enforcement of those policies on a per-message basis.

This section describes several common use cases for message-level security that is based on these capabilities.

5.3.1 Business-to-business (B2B)

The recent growth of asynchronous messaging as the global backbone of interconnected businesses has been nothing short of phenomenal. But the mainstreaming of asynchronous messaging has commoditized the skill market. Anyone today can implement a B2B messaging interface. But how many can secure it properly?

WebSphere MQ AMS provides additional protection against the risk that your business partner has failed to secure their queue manager or firewall properly. By restricting local channels and exchanging signed messages, you have much greater assurance that the messages that you consume originated from the business partner's authorized application rather than a rogue queue manager or application on their side.

A special case of the B2B interface is the clearinghouse. The topology is that of a hub and spoke where the clearinghouse serves as the hub and each spoke is a different business partner. This category includes central banks, settlement houses, insurance claims processors, retail chains, hospitality chains, service bureaus, state and local governments, and many others. When using connection security, the hub must isolate the paths for each external partner or use channel exits or other controls in order to achieve granularity of security. WebSphere MQ AMS provides better security, even while reducing the need for dedicated configurations per partner.

5.3.2 End-to-end encryption

The architecture of asynchronous messaging requires a different security model than does that of synchronous messaging. One of the primary differences is that asynchronous messaging is a custodial system. The producer of a message hands it off to the messaging network, which then persists and transports the message as required and finally delivers it to the recipient. This process opens up any number of possibilities to intercept and modify the message while in custody of the messaging provider, or even to inject new messages that did not originate from the original sender. An *enterprise service bus (ESB)* uses these capabilities to good advantage by enriching or processing the messages in some way before delivering them to the intended recipient. But the very same functionality that enables the ESB to exist can also be exploited by an attacker.

One approach to mitigate this threat is to modify the applications to sign or encrypt messages. In this way, it is possible to ensure that the messages are never tampered with while in custody of the messaging network. The components of the network have no way to read an encrypted message and any tampering of signed messages can be detected. Of course, if an ESB is used, it also must sign or encrypt the messages. Changing the applications and the ESB components can be very intrusive, complex, and expensive and lead to system instability.

Putting the encryption into the messaging network itself eliminates the expense of modifying the applications but fails to solve the problem. Since the data custodian has the capability of encrypting and decrypting the messages, the recipient has no guarantee that the message that it receives is authentic. Also, this model does not fit the definition of end-to-end encryption because the message is in plaintext while in custody of the messaging network, at the points of delivery and of consumption. Even if the messages themselves are not attacked, such systems can leak confidential data in log files, traces, and dumps and through memory inspection.

WebSphere MQ AMS addresses all of these issues by intercepting WebSphere MQ API calls and performing the cryptographic operations in the memory space of the application endpoints. The applications that are involved require little or no code changes and the data is protected before transfer of custody to the queue manager. Because the applications hold the keys, the queue manager can never render an encrypted message in plaintext so there is no danger of data exfiltration through traces, dumps, logs, or memory inspection. The messages are only in plaintext while in custody of the applications that will process them.

This use case comes up quite often in regulated environments where financial transactions, personally identifiable information, personal health information, or classified government information is exchanged. In cases where compliance is regularly audited, the cost of WebSphere MQ AMS implementation can be offset in part or in whole by the reduction in cost of the recurring compliance audits. When the queue manager does not hold keys that are capable of decrypting messages, the task of auditing the messaging nodes is greatly simplified.

5.3.3 Data aggregation

The base security model of WebSphere MQ mediates access to queues and topics based on the identity that is associated with a channel connection. But, suppose that the channel arrives from a hub or gateway carrying traffic from many different sources and each of those remote senders needs access to a different subset of queues on the local queue manager. In that case, the channel must be authorized to all possible destinations.

The applications consuming the messages can look at the message headers to perform user-based authorization but this task requires modification to the applications, embeds network security policy in application code, and assumes that the values in the message header are trustworthy.

Alternatively, WebSphere MQ channels can be configured to authorize messages based on the message header but this approach also assumes that the message headers can be trusted and, in addition, requires that the user IDs in the message header can resolve locally to the operating system. The most significant problem with these approaches is that WebSphere MQ is most efficient with a direct route between producer and consumer queue managers. Gateways and hubs are usually only used when the remote queue manager is not in the same administrative group and, by definition, the message headers are then untrusted.

In this case, WebSphere MQ AMS augments the channel security. The channel itself still must be authorized to put messages to all legitimate destinations but applications consuming the messages will filter messages based on the sender's identity as represented by an X.509 certificate. Even though the parties involved do not share administrative control of their respective queue managers, the certificate authority guarantees that no two certificates can have the same distinguished name so authorization is based on a single, shared, global namespace for identities. This enhanced authorization requires no code changes to the applications and the security policies are external and dynamically configurable.

5.3.4 Command and control flows

Requirements for strong security, including encryption, are always driven by business needs and those needs are usually represented in messaging systems by the business data. As a result, considerable attention is paid to securing that data. But if the business data is critical, it can be argued that the message flows which control the behavior of the systems hosting the data are just as critical. How useful is business data encryption if an attacker can capture the administrator's login credential exchange in plaintext? Or leach sensitive information, such as table and column names, user account names, or operational command codes from the administrative traffic flows?

Any application that listens on a queue for administrative commands is a candidate for this use case. Dashboards, administrative clients, instrumentation programs, and monitoring agents are just a few examples. Enabling a protection policy on the queue and specifying the authorized senders can provide strong authentication of administrative users. Specifying encryption in the policy renders the administrative traffic opaque to outside observers.

At least one regulatory regime has singled out administrative traffic as so important that it requires encryption in all cases. The Payment Card Industry Security Standards Council[1] sets policy and standards for the processing of payment card transactions across the globe. They require their members to encrypt all non-console administrative access using strong cryptography on any systems through which card payment data passes. But whether it is required or optional in any specific case, this often overlooked traffic should be considered when designing your security model.

[1] https://www.pcisecuritystandards.org/

6

WebSphere MQ security controls

This chapter provides a high-level overview of security controls available to protect IBM WebSphere MQ queue managers and their resources against general attack and inappropriate usage. In addition to intrusion prevention, this chapter includes controls that facilitate detection of security breaches, forensic investigation, and remediation. The intent is to provide you with a broad overview of the available controls and their potential usage and implementation. This chapter describes a palette from which to select appropriate controls to achieve specific goals and thus explore possibilities beyond the specific scenarios presented here.

Many of the controls that are mentioned are not among those controls found in the WebSphere MQ Information Center security documentation and not traditionally considered in a security context. However, these controls can be extremely useful as components in a security design. Other controls that are mentioned are external to the WebSphere MQ configuration and reside in the host server, file system, network, or other external components. In many cases, the security of WebSphere MQ depends on these external controls yet they are frequently overlooked and thus merit some consideration here. Finally, a few of the controls that are mentioned are process-oriented and policy-oriented rather than configuration controls. In the absence of rigorous process controls and policies, it is difficult to implement and maintain effective security.

This chapter includes the following topics:

- Overview
- Operating system and file system resources
- Queue manager local resources
- Channels, transmission queues, and communications
- Queues and other objects
- Applications using WebSphere MQ
- Recent changes
- Procedural considerations

6.1 Overview

Many of the controls that are presented in this chapter can be easily implemented with little effort and cost while some controls require more careful thought and planning. They address intrusion prevention, breach detection, recovery, and forensic analysis. Together, they provide a layered defense and multiple barriers that make it more difficult to perform unauthorized actions. No single installation would use all of the controls that are presented here because the cost would likely outweigh the benefits. The intention of presenting them in this format is to provide some insight into how otherwise basic configuration and tuning can impact security. You may also want to incorporate some of these controls into the scenarios that are described in this book.

Good information about securing WebSphere MQ can be found in IBM developerWorks articles, conference presentations, WebSphere MQ information centers, and other IBM Redbooks publications. This chapter is based on research from these sources and provides the key points of securing WebSphere MQ in one place. These key points should not be considered a complete list or designed to meet any particular security compliance requirements.

The scenarios in the rest of this book describe many aspects of security hardening in greater detail and how they apply to practical use cases. For this reason, Transport Layer Security (TLS)/Secure Sockets Layer (SSL) protection of channels, networking, firewalls, WebSphere MQ Internet pass-thru, gateway, multi-instance queue manager configurations, and IBM WebSphere MQ Advanced Message Security (AMS) are not discussed here.

6.2 Operating system and file system resources

This section describes basic hardening techniques that can be applied to operating system and file system controls that are external to features in WebSphere MQ.

6.2.1 File system as the root of trust in the server

Any software that is required to boot up to an operational state and run without human intervention must have access to credentials to establish its identity when communicating across the network. Frequently, these credentials are stored in configuration files as user ID and password values. Often, the credentials include private keys that are stored as X.509 certificates in keystores and it is the keystore location and password that are held in the configuration file. In either case, administrators sometimes object to storing these credentials in the configuration file in plaintext. There is limited value in obfuscating the credentials, and then only if the threat that is mitigated is well understood.

The fundamental problem is that any obfuscation of the credentials must be reversible if the application is to boot and run without any human intervention. Although this process is sometimes referred to as encryption, it is more useful to think of it as *obfuscation*. If a configuration file, or a field in that file, is encrypted, the application must have a key with which to reverse that encryption. If that key is compiled into the application, it is a straightforward process to reverse engineer the application to extract the key. If the key is external to the application, it generally resides somewhere in the file system. That secondary key could be encrypted but that would leave a tertiary key to be stored somewhere.

The exception is a *Hardware Security Module* (HSM). HSMs come in local and network versions. Since the networked version can support access from many servers, it is more cost-effective. Of course, the network HSM does not function anonymously. The servers connecting to it must supply credentials that are, once again, stored on the file system. A local HSM stores keys in tamperproof hardware. Rather than supplying the key on demand, it performs the cryptographic operations and returns the result. Unfortunately, very few servers today employ dedicated HSMs.

With the possible exception of systems that have a locally installed HSM, the root of all trust in the server is the file system. If the file system is not secure, nothing on that server is secure. Obfuscating the password protects against accidental disclosure by making the password longer and randomizing the characters. However, someone with the obfuscated version of the password can still use it to impersonate the application. Encrypting the password with a Key Encryption Key (KEK) that is obtained from the network mitigates the threat of someone stealing the disk or server but the credentials for the key server either reside on disk or are derived from something about the server. Since all of the unique configuration of the server resides on disk, ultimately the disk becomes the root of trust on the server.

Once this concept is understood, it becomes clear that file system permissions to application binary and configuration files should always be locked down. Ideally, these files will be readable only by the owner and not by their group or by the world. Furthermore, it is best if these files are read-only while the system remains operational. The administrator can temporarily change them to read/write in order to perform updates and then change them back. This process makes it more difficult for an attacker to update the configuration files through a compromised application.

For the WebSphere MQ family of products, this advice applies to passwords for key repository files. The `kdb` format that is used by queue managers and compiled MQ applications stores the password in a stash file. Although the password is obfuscated, it is easily reversed into plaintext. If the keystore files are copied as a set, no password reversal is required since the queue manager and applications will read the stash file directly.

Some WebSphere MQ components, such as file transfer agents, use Java Keystore (JKS) format keystores. These JKS format keystores do not have a standard format for obfuscating passwords but some applications provide a facility to store an obfuscated version of the password. In these cases, the keystore should still be considered unsafe since an attacker who can read the password can reverse it or copy the configuration file intact to impersonate the application.

The requirement to obfuscate passwords in configuration files is often based on an implicit assumption that this obfuscation protects against unauthorized access to the file, which is clearly not the case. Use the built-in file system permissions system as the primary protection for the configuration files. If password obfuscation is used, it is secondary and serves mainly to prevent accidental disclosure from “shoulder surfing” or other casual observation.

6.2.2 Restrict file system access

Ensure that access to WebSphere MQ installed product directories, data directories, and files is restricted to only user IDs and groups that need access. This rule particularly applies to key repository files that are used for TLS and SSL certificates and WebSphere MQ AMS certificates. Some directories and files have general access to allow any user to write, read, and execute. These permissions are set up by WebSphere MQ installation and in general they do not need to be modified because they do not present a significant security risk. It is possible to modify the permissions to be more restrictive, but this change may impede the normal operation of WebSphere MQ.

6.2.3 Restrict access to mqm home directory and tools

The WebSphere MQ administrative account requires a home directory in which certain configuration files are stored. For example, the WebSphere MQ AMS `keystore.conf` file and keystore are located here, by default. Non-administrative users should have no access to read, write, or execute files in this directory. This restriction includes the home space for the mqm account on UNIX and Linux systems and the platform equivalent on others.

On UNIX and Linux systems, the WebSphere MQ installer will create the mqm account if it does not already exist. If the account is created by the installer, it will have `/var/mqm/` set as its home directory. Although this approach is convenient for the user, it leaves the mqm home space located in a directory to which ordinary users must have some level of access. For this reason, create the mqm account in advance and specify a private home space directory.

6.2.4 Limit access to the mqm user ID

On UNIX platforms, the mqm user ID provides full administration control of WebSphere MQ processes and access to all internal files and queued messages. Usage of this user ID should be restricted to legitimate purposes that are required by WebSphere MQ operations and support procedures, such as starting and stopping processes, running some command programs, managing WebSphere MQ objects and resources, and file and directory manipulation. If a person needs to use the power of the mqm user ID, their actions should be accountable and auditable.

It is preferable to set the password of mqm to an unusable value, or deny logon and remote access rights. Local access to this user ID should still be allowed and controlled by the **`sudo su - mqm`** command or other similar user ID switching capability. The **`sudo`** command should be configured to require the user to enter their personal password. This requirement will prevent someone from casually using a terminal session while a WebSphere MQ administrator is not at their workstation and switching to the mqm user ID.

In this case, the mqm password is not required to be known, yet all possible actions under this user ID can be performed, and the actions are accountable to an individual.

WebSphere MQ operations should not be performed from a root or similar system administrator user ID. The root user should switch to the mqm user ID and then perform the command, for example:

```
sudo su - mqm -c strmqm MYQMGR
```

Starting or running WebSphere MQ internal processes directly from root is not recommended because it gives started processes elevated authority that is not required and may actually cause problems.

These same principles and practices apply to user IDs that are required by WebSphere MQ product installation and operation on other platforms.

6.2.5 mqm group membership

User IDs that are members of the mqm group have wide ranging administrative powers in WebSphere MQ and full access to all its resources, including objects in queue managers. Therefore, membership of this group should be restricted to WebSphere MQ operational and support staff. Developer teams, application support teams, and application program user IDs should not be members of this group in any environment, including development, test, and production.

New objects can be defined in queue managers by the mqm user ID or other user IDs that are members of the mqm group, for example, by using the **runmqsc** command. On UNIX platforms, ensure that the user ID's primary group is mqm, and not some other group, such as staff or users. The reason is that WebSphere MQ creates an object authority manager (OAM) security profile for each newly defined object with full authority to the user ID's primary group. When the user's ID belongs to a primary group other than mqm, full authority over the new object is granted to all other user IDs in that group.

This side effect is not usually desirable, especially if the user ID's primary group was staff or users, since this grants every user ID on the system full authority to the new object. This situation can create security compliance issues that will need to be remediated. This issue is further discussed in 4.11.3, "Administrative users with mqm as a secondary group" on page 31.

6.2.6 Files and directories

WebSphere MQ is only as secure as the security of the underlying file systems and infrastructure, such as NFS or SAN facilities. If file and directory permissions are modified or root access to the operating system is compromised by unauthorized access, there can be no guarantee that WebSphere MQ remains secure.

Disk input/output channels and communication buses should also be highly secured. WebSphere MQ input/output on queue files and recovery log files exposes information about application messages in the clear, unless WebSphere MQ AMS is used. Network sniffers, bus monitors, and other snooping tools should be tightly controlled.

WebSphere MQ installation is performed by root or an administrator user ID and it creates directories and files with very specific access permissions for the mqm user ID and the mqm group. There are also directories and files that can be read or written by any user on the system. These permission settings may seem to be security weaknesses but they are required for normal operation of WebSphere MQ. For example, WebSphere MQ interface libraries that are bound to application processes may need to write to the error log files or generate IBM First Failure Support Technology™ (FFST™) files. If access permissions on these files are changed, it may cause problems.

6.2.7 Fully specified names in mqm cron job scheduler

On UNIX platforms, ensure that all entries in the cron table for the mqm user ID do not refer to any programs, scripts, directories, or files in unprotected directories. This list includes mounted directories that may be accessible from other systems with different protections. All directories and file names should be specified in full so that they do not rely on the mqm account's **PATH** or any other settings in its shell logon script. The referred to directory trees should not contain any general permissions that would allow an attacker to modify or create any files that could interfere with the operation of scheduled jobs. The same principles apply to other platforms.

6.2.8 Do not administer WebSphere MQ as root

The root user ID on UNIX, or equivalent super user system administrator capabilities on other platforms, should not be used to perform WebSphere MQ administrative actions. Doing so may result in programs running with higher levels of authority than they normally would and may provide avenues for an attacker to exploit this elevated level of authority. Normally, only WebSphere MQ installation tasks require root access.

6.2.9 Protection of WebSphere MQ backups

WebSphere MQ security is only as strong as the security of its underlying file system, and the same rule goes for backups of the file system. Backup processes and media should be protected with appropriate security controls so that unauthorized programs and people do not have access to view, restore, or copy any WebSphere MQ backups.

6.2.10 Increase the size of error logs

By default, each queue manager has three error log files of up to 2 MB each in size for WebSphere MQ V7.1 and later or up to 256 KB for WebSphere MQ version 7.0 and earlier. These files contain a limited number of the most recent error and information messages. In a case of sustained attack or continuous security failures, the logs may roll over very quickly. Historical information about what happened immediately before the failures may be completely lost.

It is possible to increase the size of the error log files by editing the `qm.ini` file for a queue manager. For further information, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa12710_.htm

There are also error log files that are scoped globally and not associated with specific queue managers. When an error occurs at a point before the target queue manager is identified, the error is recorded here. For example, if the **dspmqver** command fails, that error is recorded in the global error log files.

These global logs reside in `{WebSphere MQ install}/errors` on distributed platforms and can be written to by any WebSphere MQ application program. The maximum size can be increased in WebSphere MQ version 6.0 and later by setting **MQMAXERRORLOGSIZE**, a global environment variable, in the shell of all users. For further details, see the following IBM Technotes:

http://www.ibm.com/support/docview.wss?uid=swg21179310

6.2.11 Archiving error logs

The queue manager maintains a rolling set of log files such that `AMQERR01.LOG` is always the latest and `AMQERR03.LOG` is always the oldest. When `AMQERR01.LOG` fills, the queue manager executes the following procedure:

- Delete `AMQERR03.LOG`.
- Rename `AMQERR02.LOG` to `AMQERR03.LOG`.
- Rename `AMQERR01.LOG` to `AMQERR02.LOG`.
- Write the new entry to a new copy of `AMQERR01.LOG`.

The time window that is represented by the information in the error logs at any given moment depends on their size and the level of activity of the queue manager. An additional audit trail may be captured by backing up or archiving `AMQERR03.LOG` whenever its timestamp changes. Since this value only changes during a log rotation event, monitor the timestamps at intervals that are sufficiently short to guarantee that the logs will not rotate more than once during each polling interval.

6.2.12 Isolation of staging environments

Most applications will have several environments for lifecycle staging, such as development, unit test, integration test, quality assurance, pre-production, and production. Not only should these environments each use different WebSphere MQ queue managers, they should be located on different server instances. Port numbers, channel names, and queue name qualifiers should also be different. Queue manager names should indicate the purpose of the environment, such as including a single letter, such as “D” for development, “T” for test, or “P” for production.

This approach provides a shield against unauthorized or accidental transfer of data between environments. A security breach in one environment is less likely to impact another.

To facilitate such isolation, applications should be designed to make it relatively easy to move execution from one environment to another. WebSphere MQ configuration details, such as channel name, host, port number, and queue object names, should be dynamically obtained from a configuration file, directory service, or name service, based on the environment in which it is executing. The configuration details should be secured against unauthorized access. Names should not be hardcoded into application programs. WebSphere MQ client applications should be designed without a dependency on the queue manager name.

6.2.13 Protect user-provided executables

WebSphere MQ provides the capability to execute arbitrary operating system commands and programs based on SERVICE and PROCESS object definitions. Programs and scripts that are executed by the queue manager run with the queue manager’s authority so an attacker who can alter these files can substitute malicious commands and gain control of the queue manager’s account. Operating system commands are usually restricted so that only the root or system administrator can change these files. Ensure that any scripts or executables referred to by SERVICE or PROCESS objects are similarly protected from changes.

6.3 Queue manager local resources

This section describes basic hardening techniques that can be applied to general resources that are managed by queue managers. Later sections in this chapter apply to specific queue manager resources, such as channels, queues, other objects, and applications.

6.3.1 Define a system dead letter queue

When creating a queue manager, a dead-letter queue (DLQ) should be assigned so that undeliverable messages can be requeued, for example:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE MYQMGR1
```

Ensure that the maximum message length (MAXMSGL) is large enough to cater for the largest possible message that the queue manager and incoming channels will handle, plus at least 172 bytes for a dead letter header (MQDLH) structure. MAXMSGL defaults to only 4 MB.

The maximum depth (MAXDEPTH) defaults to 999,999,999 messages. This value can be reduced to a lower number as a compromise between being able to store all legitimate messages that cannot be delivered until they can be reprocessed and reducing the impact of a resource exhaustion attack.

It is possible to assign or change the DLQ after the queue manager has been created, for example, by using the MQSC command:

```
ALTER QMGR DEADQ('SYSTEM.DEAD.LETTER.QUEUE')
```

All Message Channel Agents (MCAs) and trigger monitors should have access to put messages to the DLQ, while access to get messages should be restricted to the DLQ handler process (see 6.3.2, "Considerations for dead-letter queue handler" on page 48).

In WebSphere MQ version 7.1 and later the USEDLQ attribute can be set on distributed and cluster channels to control whether the DLQ is used for messages that cannot be delivered. This attribute should be left at its default value (YES) so that these messages can be recorded. For the alternative value (NO), the MCA will follow the normal procedure for dealing with channel problems, but it will not use the DLQ.

In general, application programs should not be able to directly put messages to this queue. Applications should have dedicated queues for this purpose, for example, `APP1.DEAD.LETTER`. The application can add a standard MQDLH structure to the beginning of the message data to make it possible for a dedicated DLQ handler process to take action on the message. This approach reduces the exposure to abuse of the system DLQ by flooding the queue with application messages, or by inappropriate inspection, modification, or redelivery of messages.

An application DLQ has a different purpose than an application Back Out Queue (BOQ). A BOQ contains poison messages that repeatedly fail to be committed in an application Unit of Work (UOW). An application DLQ should contain messages that are bad or erroneous for other reasons.

There are some situations, such as a gateway queue manager in a business-to-business (B2B) design, where a DLQ may not be desirable. In B2B designs, static routing provides an additional security control that isolates the namespaces of the multiple business partners that are involved and is often enforced by design. Given this constraint, it should not be possible to receive a mis-addressed message from a business partner, and the receipt of one indicates a problem of some sort, or possibly an attack. In this case, it is often desirable for mis-addressed messages to result in a channel outage rather than to stack up in the DLQ. For an example, see 10.3.2, "Gateway queue manager configuration" on page 155.

6.3.2 Considerations for dead-letter queue handler

The purpose of a DLQ handler is to retry messages that land on the DLQ for transient errors. For example, if a destination queue is full when a message arrives, the message may land on the dead letter queue. Often, such errors are caused by a temporary spike in message production, and the application servicing the target queue eventually catches up. As the queue begins to drain, it is useful to requeue any messages that spilled over into the DLQ so that they may be processed.

The queue manager creates a new message, attaches a DLQ header, and then appends the entire original message, including all of its headers, to the end. The DLQ handler is then able to re-create the original message exactly as it was sent by setting all of the message header fields as it writes to the target queue. In order to perform this function, the DLQ handler must run with elevated authority. Attackers may attempt to exploit this elevated authority by crafting messages with appropriate DLQ headers and addressing them directly to the DLQ. The DLQ handler will then strip the DLQ header and write the malicious message to the attacker's chosen destination queue, using the authority of the DLQ handler.

To mitigate this threat, begin by restricting access to the DLQ. As a rule, applications should use application-specific exception queues rather than the system DLQ. If this restriction is enforced, only inbound receiver, requester, and cluster receiver channels will ever put messages to the DLQ.

The DLQ handler is described in detail in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa15920_.htm

Running the DLQ handler under the same account as the channels

One of the basic security controls that prevents the escalation of low-privileged accounts to administrative privileges is to run all requester, receiver, and cluster receiver channels under an MCAUSER account that is restricted to a subset of queues. If the DLQ handler is run under the same account as the channels, it will not be possible for messages to land on any queues other than the queues to which the channels are already authorized.

This method works best when all inbound channels use a common MCAUSER because the DLQ is a system-wide resource that is shared among all the channels. When the channels run under multiple MCAUSER accounts, the DLQ handler becomes a choke point where the granularity of channel authorizations is reduced. This subject is discussed further in the next topic.

DLQ handler and granular channel security

Some queue managers where more granular authorizations are required use different MCAUSER accounts for each inbound channel. Because there is only one DLQ per queue manager, it is not possible for the DLQ handler to determine which channel delivered a particular message and apply the appropriate authorization check. Since the DLQ handler cannot provide the same level of granularity as the channel MCAUSER values, in this case, running the DLQ handler actually reduces granularity of the channel security.

The question of whether the DLQ handler can be run on a queue manager with granular channel authorizations depends in large part on whether the adjacent queue managers are managed by the same administrative team as the local queue managers. If so, it is possible for the administrators to enforce the granularity through strong authentication on the remote queue managers and to then monitor those configurations to ensure compliance. If the remote queue manager's configuration is trusted and verifiable, this can compensate for the reduced granularity at the receiving queue manager.

However, when the adjacent queue managers are administered by a different team, it would be wise to restrict those channels from placing messages onto the DLQ. Prior versions of WebSphere MQ accomplished this by restricting authorizations of the channel's MCAUSER account. Newer queue managers can utilize the channel's USEDLQ attribute to control this behavior. It is advisable to use both of these controls in combination where a *defense-in-depth* approach is required.

DLQ handler and B2B

In the case of queue managers that host B2B channels, the security risk that is represented by the DLQ handler exceeds the benefit. Preferably, such queue managers do not have a defined DLQ, and any undeliverable messages cause the channel that is transporting the message to stop. However, in the case that a B2B queue manager has a defined DLQ, monitor the depth of the DLQ rather than automating dead message handling.

General DLQ handler considerations

The DLQ handler that is supplied with WebSphere MQ is configured with rules that describe to it which errors to process and how to dispose of messages in each category. Consider carefully the impact of the rules that govern the DLQ handler's behavior.

The DLQ handler should not specify the redelivery of messages that were placed on the DLQ because of an authorization failure (MQRC_NOT_AUTHORIZED). These failures should be forwarded to a dedicated queue, manually investigated, and resolved.

When dynamic Reply-To queues are used, messages often land on the DLQ with a reason code indicating the target queue does not exist. It is common practice to configure the DLQ handler to delete such messages. Consider though that an attacker attempting to enumerate queue names can use this configuration to their advantage because any messages that are sent to a non-existent queue are silently deleted by the DLQ handler. To mitigate this threat, ensure that all dynamic queues use a known name prefix and configure the DLQ handler rules to only delete messages addressed to queues with that prefix and that also have the MQRC_UNKNOWN_OBJECT reason code.

Consider requeuing dead messages to one or more secondary dead letter queues based on some criteria, such as the destination queue or the reason code. These messages can then be adjudicated manually or by using a secondary DLQ handler instance. This approach keeps the system DLQ empty so that it can be monitored and alarmed based on non-zero depth.

Do not use rules that request the handler to put messages using the context user ID of the message. This user ID can be easily set to any value by an untrusted remote queue manager or application.

All DLQ handler rules should be carefully reviewed and tested before implementation to ensure that the handler is working as intended.

6.3.3 Enable event messages

A queue manager can generate instrumentation messages for a wide range of events. It puts them onto system event queues and does not process them further. When event messages are enabled, the administrator must arrange for a program to monitor the event queues and take appropriate action on each message. The monitoring application can be continuously running, triggered, or run at scheduled times to process all the messages. If this is not done, the event queues will become full of old event messages and new event messages will be dropped. There are many commercial software products that provide WebSphere MQ event monitoring and alerting capabilities, such as IBM Tivoli OMEGAMON XE for Messaging.

Events and monitoring are described in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/mo10380_.htm

The Information Center contains a summary of all events:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/mo10630_.htm

The event queues are local queue type objects by default, but they can be deleted and redefined as remote queues that transport event messages to a remote queue manager collection point. This depends on the reliability of the WebSphere MQ channel and network infrastructure.

With publish/subscribe (pub/sub) capability built into all queue managers now, the event queues can also be defined as aliases to topics within a system/event/ topic tree. This approach allows many applications, which might be interested in the event messages, to receive them either by subscribing directly or by reading a queue that is the target of an administrative subscription.

Events are also discussed in 2.3.3, “Monitoring and accountability” on page 9.

6.3.4 Restrict access to remote clustered queues

WebSphere MQ version 7.1 and later provides the ability to enforce authorization checks against non-local clustered queues. Prior versions required the administrator to grant access to the cluster transmit queue, which in turn allowed the application to address messages to any remote destination in the cluster. Alternatively, the administrator could define QREMOTE or QALIAS objects on the local queue manager pointing to remote queues and then authorize the local objects. This approach provides the granularity required but at the expense of forcing the administrator to create a local object for every legitimate remote destination.

By using the new functionality, the administrator can restrict access to the cluster transmit queue and instead grant applications access to specific queues advertised to the cluster. For further details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/qc11430_.htm

6.3.5 Do not disable WebSphere MQ authorization checks

WebSphere MQ is packaged with an authorization service component named OAM on UNIX, i5/OS, and Microsoft Windows platforms. On the z/OS platform, the queue manager calls System Authorization Facility (SAF), which invokes an External Security Manager (ESM), such as IBM RACF®, TopSecret, or ACF2 to service authorization requests. By default, this component is enabled in all versions of WebSphere MQ.

Do not disable authorization checking since disabling it will remove nearly all points of security and expose a myriad of opportunities for attack and misuse of the product. This rule applies to all application environments, including development.

To ensure that authorization checking remains active, consider the following configuration:

- Do not set the environment variable **MQSNOAUT=**YES when creating a queue manager on UNIX or Windows platforms.
- Do not remove or change values in the Service stanza or ServiceComponent stanza in a queue manager’s qm.ini file on UNIX or Windows platforms. Prior to WebSphere MQ version 7.1 on Windows platforms, this stanza was stored in the Windows Registry. The exception is when a custom authorization service has been installed.
- On z/OS, do not define switch profile names that begin with NO. These names disable categories of security checking on the z/OS platform.
- On z/OS, do not define hlq.RESLEVEL profiles. These profiles bypass all resource security checks for a queue manager or queue sharing group on the z/OS platform.

After security has been disabled, there is no supported procedure to reenable it other than to back up the queue manager’s objects, delete the queue manager, redefine it, and restore the objects.

6.3.6 Generic authorization profile names

WebSphere MQ object security profiles can be defined to protect access to specific objects, or a generic group of objects that have a common pattern in their name. It is tempting to use just a few very generic profiles as it reduces the number of profiles and simplifies the configuration. However, it does not always provide the minimum authorities that are required by each group to each object. This practice makes it possible for a group to access queues and messages to which it may not be properly entitled in the design of the application, and this ability opens up attack vectors and opportunities for misuse.

For example, a set of local queues for an application suite all have names beginning with `MYAPP1`. These queues are processed by several different programs; some are request queues and some are reply queues. A generic profile `MYAPP1.**` could be easily defined with `put` and `get` authority to all groups that are used by the applications, but this approach provides more privileges than are actually needed and reduces the effective level of security.

As an alternative, define at least two profiles for each object, such as `MYAPP1.PAY.SALARY.UPDATE.REQ` with put authority granted to the group that makes requests and get authority granted to the group that processes the requests and updates the payroll database, and two corresponding profiles for `MYAPP1.PAY.SALARY.UPDATE.REP`. There may also be queues for the pay processing applications to update or inquire about employee address or banking details. Additional capabilities of generic profiles can be used to implement the minimum required security, yet keep the profiles to a manageable number, for example, `MYAPP1.PAY.*.REQ` and `MYAPP1.PAY.*.REP`, or even `MYAPP1.**.REQ` and `MYAPP1.**.REP`.

6.3.7 PROCESS and SERVICE objects should use explicit paths

WebSphere MQ PROCESS and SERVICE objects enable a queue manager to run arbitrary programs on demand or based on the arrival of messages in a queue. When defining these objects, all directories and file names should be specified in full. The use of relative path names introduces the risk of a path traversal attack. The use of unqualified program names introduces a dependency on the queue manager's **`PATH`** environment variable.

6.3.8 Run the command server only when it is needed

The WebSphere MQ command server is a long running process that gets Programmable Command Format (PCF) messages from the system command input queue, executes them with WebSphere MQ administrator rights, and puts the output of the command as reply messages to the nominated reply queue. This feature is very powerful and can be used by an attacker to obtain elevated access to the queue manager or to execute arbitrary operating system commands with the queue manager's administrative user ID.

The primary security control over the command server is the use of authorization profiles to restrict access to place messages onto the command queue. Since non-administrators have no access by default to any queues or other WebSphere MQ objects, the administrator will have made a conscious decision to create authorization profiles to allow this level of access. Authorization profiles remain the primary security control for the command server but only address risks from non-administrative users.

In some scenarios, it is necessary to consider the risks of accidental unplanned changes or the risk of an attacker gaining elevated access to the queue manager. In these cases, a simple mitigation strategy is to run the command server only during authorized change windows. Although the command server is started by the queue manager by default, this is configurable and it can be ended by using the **endmqcsv QMGRNAME** command with optional parameters **-c** for controlled and **-i** for immediate. It can be started on demand by using the **strmqcsv QMGRNAME** command with optional parameter **-a** to block PCF commands for authority record objects. The **dspmqcsv QMGRNAME** command displays its current status.

If this approach is taken, be sure to clear the system command input queue before starting the command server in order to delete any pending commands that may have been placed on the queue by legitimate users or by an attacker. One design is to create a script named `strmqm`, which verifies that there are no input or output handles on the queue, clears the queue, and then checks that the depth is zero before calling the native **strmqm** command.

When using this security control, remember its limitations. There is a window of time between the clearing of the command queue and shutting down the command server during which an attacker can still submit commands. Selectively running the command server during authorized change periods reduces the attacker's window dramatically but does not eliminate it totally.

The command server is required to run desktop tools, such as WebSphere MQ Explorer, and monitoring tools, such as IBM Tivoli OMEGAMON XE for Messaging. While the command server is shut down, these components will not provide full function or may not work at all.

6.3.9 Limited use of trigger monitors

WebSphere MQ includes a trigger monitor program runmqtrm that is designed to execute operating system commands that are based on the arrival of a message in a triggered queue. When trigger conditions are met, the queue manager puts a special message to an initiation queue that is monitored by the trigger monitor program. The message contains the name of the queue that generated the trigger and the command to be executed by the trigger monitor program. This design allows a single trigger monitor instance to serve many triggered queues. The trigger monitor then executes the command that it received in the trigger message and passes the name of the queue and any other parameters that are specified.

Normally, the queue manager obtains these details from a process object, which has been defined by the WebSphere MQ administrator. If an attacker can define or alter the process definition, it is possible to cause the trigger monitor to execute any arbitrary command. A more direct attack is possible if the attacker can place messages directly onto the initiation queue. By crafting the message to look like a genuine trigger message, the attacker can execute programs against any queue, even those queues that are not configured for triggering. Furthermore, there is no constraint that the programs that are executed must interact with the queue manager at all. The executed command could, for example, copy, edit, or delete files.

By default, the queue manager does not run a trigger monitor, but there is a queue named `SYSTEM.DEFAULT.INITIATION.QUEUE`, which is defined when a queue manager is created. The trigger monitor that is provided with WebSphere MQ will use this queue by default but that behavior may be overridden with the **-q** command-line switch. Since the system initiation queue name is well known, an attacker may blindly send messages there in an attempt to execute commands. For this reason, where triggering is used, configure custom initiation queues to which only the queue manager and administrative accounts can put messages, and from which only the account running the trigger monitor should be able to get messages.

Another control that may be employed is to run the trigger monitor from a low-privileged user ID. This account would be the only account that is authorized to get messages from the initiation queue, and strict operating system controls can be implemented to restrict the commands that it can execute.

In a default installation of WebSphere MQ version 7.5 on most platforms, `runmqtrm` execute permissions are restricted to WebSphere MQ administrative accounts. To run a trigger monitor from a non-administrative user ID, the WebSphere MQ administrator should copy it to another directory that is controlled by the WebSphere MQ administration team and grant minimal permissions to an application group to execute the program. The application group requires these authorities:

- Connect and inquire authority to the queue manager
- Get authority to the initiation queue
- Put and passall authority to the queue manager's DLQ

The runmqtrm trigger monitor connects using bindings mode and executes commands on the same server that hosts the queue manager. Another version of this program called runmqtrmc exhibits the same behavior except that it runs as a WebSphere MQ client and executes commands on any host that is supported by WebSphere MQ clients and with network access to the queue manager. This trigger monitor would typically use the same SVRCONN channel and authentication methods as the application that it serves.

There is nothing special about trigger monitors that are supplied with WebSphere MQ. They do not use internal API calls or protected methods. It is entirely possible to write a trigger monitor in C, Java, or any of the other supported languages. One reason to write a custom trigger monitor would be to enhance the security of the program further. Because the trigger monitor that is supplied with WebSphere MQ is intended to be a general-purpose tool, it is designed to execute any arbitrary command that is passed to it. This design makes it difficult to restrict the operation of the trigger monitor, especially a trigger monitor that runs with the WebSphere MQ administrative account. In high security use cases, a custom trigger monitor that executed only commands from an enumerated list would allow the administrator some configuration latitude without allowing an attacker unrestricted access to all commands.

There is a special purpose trigger monitor program runmqchi that is used to start channels from trigger messages put to the `SYSTEM.CHANNEL.INITQ`, as a result of messages being put to transmission queues. This program is considered part of the queue manager and does not require any specific treatment for security.

6.3.10 Minimal authority on SYSTEM objects

A large number of MQ objects are automatically defined when a queue manager is created. They all have names beginning in `SYSTEM.` and are used for a wide variety of purposes, such as management of channel synchronization, storage of OAM profiles, event message queuing, and as templates for the creation of new objects.

Most `SYSTEM.` objects are fully functional. For example, `SYSTEM.DEFAULT.LOCAL.QUEUE` acts like any other local queue and the inbound `SYSTEM.*` receiver, requester, cluster receiver, and SVRCONN channel objects all accept connections from remote queue managers or clients. By default, only the MQ administrator has full access to these objects due to its `mqm` group membership. Consider carefully the security objectives before granting access to these objects to low-privileged principals and groups.

There are minimum requirements for SYSTEM object security on the z/OS platform to use CSQUTIL, the ISPF panels, and the Channel Initiator for distributed queuing. These requirements are described in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs12340_.htm

There are two types of authorization to consider for SYSTEM objects. *API authorizations* are those authorizations that are executed by programs directly against the object. API authorizations include `put`, `get`, `browse`, `inquire`, and a few others. *Command authorizations* apply to command server requests. These authorizations include `display`, `control`, and `controlx`. The two types of authorization may be granted independently of each other.

Some applications and WebSphere MQ components require access to subsets of the SYSTEM queues. Examples include IBM WebSphere MQ File Transfer Edition agents and tools, as well as IBM WebSphere Message Broker Toolkit. Be careful when authorizing users of these applications to grant authorities to only those SYSTEM queues that are actually required.

6.3.11 Object names

When devising a naming standard for WebSphere MQ objects, an inverse relationship exists between security and ease of administration. Well-structured object names that reflect characteristics of the system or application are intuitively easy for the administrator to work with. For example, embedding a business partner's account number in their queue name allows the administrator to easily track back to the business partner in case of a problem. However, it also provides an attacker a means to enumerate the queues by iterating through all possible account numbers. Similarly, leaving the well-known system objects in their functional default state provides a foothold for an attacker.

Where security requirements are high, consider whether knowledge of one object name would reveal the naming convention or provide clues for an attacker to derive additional object names. If so, balance the potential benefit of using unique, unrelated, less structured names against the additional administrative overhead that these names would incur. At the very least, it should not be very easy for an attacker to guess or generate valid object names without access to internal documents or business knowledge of the organization.

B2B interfaces are a special case for naming. One of the important functions of a gateway queue manager is to isolate the internal namespace from the outside world. The proper use of queue manager aliases, QREMOTE objects, and QALIAS objects provides the business partner with a set of externally visible names without exposing details of the internal network. One approach is to implement well structured and easily administered names on the internal network but to expose unstructured and opaque names to external parties on the gateway queue manager.

When considering this security control, remember that obscure naming can be costly to implement and more difficult to diagnose and resolve problems. It is therefore only justifiable in high-risk cases.

6.3.12 Realistic attribute values

The default values of attributes in WebSphere MQ are designed to support the majority of applications. Often, these values will be tuned by the administrator to enhance performance, reliability, or both. However, settings that are too permissive can impact the security of the queue manager.

To illustrate the issue, consider an example where an application attempts to put a message onto a queue and receives a return code indicating that the queue is full, treats this return code as fatal, and then shuts down. The developer sees that the queue's MAXDEPTH is set to the default of 5,000 and requests that the WebSphere MQ administrator increase the value to the maximum, which is 999,999,999 or just short of one trillion messages. The application is restarted and continues to run past the previous point of failure.

Although this action solves the immediate problem, it creates a potentially worse one. In the example, the application exceeded an artificial limit imposed by the MAXDEPTH tuning parameter. Although the application crashed, better error handling could allow the application to treat the error as transient and periodically retry putting the message to the queue. At worst, the affected queue and associated applications would become unavailable.

However, setting the queue's MAXDEPTH to the maximum value results in a situation where the potential storage that is required for the queue is approximately 3.7 petabytes. For most installations, this value will exceed the actual disk storage allocation for queue files. The result is that instead of hitting a soft limit, resulting in a transient error or outage of a single queue, the application receives no error or warning until all disk space that is allocated to queues has been consumed, thus causing a system-wide outage that affects the queue manager and all applications connected to it. It is much better for an application to encounter a recoverable soft error than an unrecoverable resource exhaustion error. For this reason, it is advisable to consider carefully the tuning values that are used by applications and system programs.

This consideration also extends to default system objects. For example, by default `SYSTEM.DEAD.LETTER.QUEUE` is set to `MAXDEPTH(999999999)` rather than the more conservative value of 5000 that is used by most other queues. Because this default is well known, an attacker may attempt to exploit the setting by flooding that queue with many large messages. To mitigate this potential exposure, the administrator might choose to reduce MAXDEPTH on `SYSTEM.DEAD.LETTER.QUEUE` to a more reasonable value.

Attributes which potentially affect security and should be reviewed include:

- MAXMSGL, MAXINST, MAXINSTC, and SHARECNV on channel objects
- MAXMSGL and MAXDEPTH on transmission queues, system queues, and application queues
- MAXHANDS, MAXUMSGS, and MAXPROPL on queue managers
- MAXCHANNELS and MAXACTIVECHANNELS in the channel stanza of the `qm.ini` file

Recommending specific values is beyond the scope of this book and depends on your business requirements.

6.4 Channels, transmission queues, and communications

This section describes security controls that are associated with WebSphere MQ channels. These techniques do not block all unauthorized access or provide complete security controls, but they are an essential part of layered defense.

6.4.1 Use channel authentication rules

WebSphere MQ version 7.1 introduced channel authentication rules (CHLAUTH) that can provide several additional controls on the initial connection of incoming channels from remote queue managers and clients. By default, there is no remote access allowed to SYSTEM channels, and all administrative user IDs are blocked on client channels. Therefore, the administrator must explicitly define new CHLAUTH rules in order to provision remote administrative access.

CHLAUTH rules should be used in conjunction with TLS/SSL to implement rigorous authentication and authorization of the identity of remote parties at the level of network IP address, remote queue manager name, presented user ID, and SSL distinguished name.

CHLAUTH is used extensively by the scenarios in this book. Additional details of this feature are presented in Chapter 3, “Authentication and authorization” on page 13 and Chapter 4, “Connection-level security” on page 21.

6.4.2 Disable all incoming SYSTEM channels

Channel objects that have names beginning with `SYSTEM.DEF.*` are designed to be used as templates for defining new channels so that parameters that are not specified in the MQSC definition command can take default values. There are also `SYSTEM.AUTO.*` channels that are used to dynamically define SVRCONN and RECEIVER type channels when automatic definition is enabled on the queue manager.

With WebSphere MQ V7.1 and higher, the default CHLAUTH rules prevent access to inbound SYSTEM channels. On queue managers where CHLAUTH is disabled or on versions prior to V7.1, the administrator must take steps to disable these channels. If such steps are neglected, the channels allow remote administrative access and the ability to execute arbitrary system commands with the WebSphere MQ administrative user ID.

For example, an attacker with access to WebSphere MQ server software can create a queue manager, delete its `SYSTEM.DEF.RECEIVER`, and define a Sender instance with that name. This SENDER channel can then connect to the default RECEIVER on the local queue manager and can be used to transmit messages. The SVRCONN channels `SYSTEM.DEF.SVRCONN`, `SYSTEM.ADMIN.SVRCONN`, and `SYSTEM.AUTO.SVRCONN` are also well-known points of entry that can be exploited by remote client applications.

It is a simple matter to disable all system channels that can be used for incoming connection requests. Several channel attributes can be utilized to provide barriers at different points in WebSphere MQ channel processing. One or more of these channel attributes can be used to equal effect:

- MCAUSER can be set to an invalid value or a user ID that does not exist on the local system, for example, `MCAUSER('*NOACCESS')`. Do not use a blank value or an administrator user ID, which may result in the channel asserting an MCA user ID that has elevated privileges on the local queue manager.

  The usage of *NOACCESS is described in the WebSphere MQ Information Center:

  http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs14190_.htm

- MAXMSGL can be set to a value less than the length of the Message Descriptor so that no messages can be transmitted on the channel, for example, `MAXMSGL(1)`.
- SCYEXIT can be set to an exit module that does not exist or a custom-written exit module that exists and is coded to stop the channel instance from running, for example, SCYEXIT(‘noexit’) or SCYEXIT(‘stopchl.so(entry)’).

Example 6-1 provides example MQSC commands that would prevent unauthorized usage of these channel names.

Example 6-1 MQSC to disable incoming system channels

```
ALTER CHANNEL('SYSTEM.AUTO.RECEIVER') CHLTYPE(RCVR) +
  MCAUSER('*NOACCESS') MAXMSGL(1)
ALTER CHANNEL('SYSTEM.AUTO.SVRCONN') CHLTYPE(SVRCONN) +
  MCAUSER('*NOACCESS') MAXMSGL(1)
ALTER CHANNEL('SYSTEM.DEF.CLUSRCVR') CHLTYPE(CLUSRCVR) +
  MCAUSER('*NOACCESS') MAXMSGL(1)
ALTER CHANNEL('SYSTEM.DEF.RECEIVER') CHLTYPE(RCVR) +
  MCAUSER('*NOACCESS') MAXMSGL(1)
ALTER CHANNEL('SYSTEM.DEF.REQUESTER') CHLTYPE(RQSTR) +
  MCAUSER('*NOACCESS') MAXMSGL(1)
ALTER CHANNEL('SYSTEM.DEF.SERVER') CHLTYPE(SVR) +
  MCAUSER('*NOACCESS') MAXMSGL(1)
ALTER CHANNEL('SYSTEM.DEF.SVRCONN') CHLTYPE(SVRCONN) +
  MCAUSER('*NOACCESS') MAXMSGL(1)
```

The `SYSTEM.ADMIN.SVRCONN` channel is not one of the default channels defined when the queue manager is created but is commonly defined by administrators for use with WebSphere MQ Explorer. This channel should either be disabled like all other inbound SYSTEM channels or properly secured with CHLAUTH profiles and TLS/SSL.

Disable the default SYSTEM channels but do not delete them. Deleting them requires any new object definitions to specify every possible attribute for that object type and this requirement presents more of a hardship to routine administration than it does to an attacker. After the default objects are deleted, they can be re-created by using the **-c** parameter of the **strmqm** command. Be aware that this option does not merely fill in any missing objects but actually overwrites any existing objects with the default values and will erase any customizations. Also, be aware that although **strmqm -c** re-creates objects, such as channels and queues, the default CHLAUTH rules are *not* created or restored by the command.

In WebSphere MQ version 7.1 and later, the default channel authorization rule `SYSTEM.*` blocks all access from all IP addresses. The system channels should still be disabled by using other methods, such as `MCAUSER('*NOACCESS')` and `MAXMSGL(1)` to provide an additional layer of protection. This approach guards against the intentional or inadvertent disablement of the CHLAUTH rules.

Preferred practice: When defining new incoming channels by using MQSC commands, be sure to specify valid values for all parameters that have been set to unusable values on the corresponding system default channel. For example, always set MCAUSER, MAXMSGL, and SCYEXIT:

```
DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(RCVR) TRPTYPE(TCP) +
  MCAUSER('qm1mcausr') MAXMSGL(4194304) SCYEXIT(' ') +
  DESCR('Receiver from QM1') +
  Other parameters...
```

6.4.3 Always specify a low-privileged MCAUSER

When defining a new inbound channel (those channels of type RCVR, RQSTR, CLUSRCVR, or SVRCONN), always specify the MCAUSER parameter. This parameter sets the effective user ID of the Message Channel Agent (MCA) process, which is then used for authorization checks. The exact behavior depends on the PUTAUT attribute setting on the channel, CHLAUTH profiles, TLS/SSL certificates, and any security exits installed on the channel. For further information, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy10790_.htm

If the MCAUSER parameter is not explicitly specified when the channel is defined, the value inherits from the corresponding SYSTEM default channel object of the same type. If the SYSTEM object contains a blank MCAUSER, the new channel's MCAUSER will also be blank, which may permit administrative access. If the SYSTEM object contains an MCAUSER value that disables the channel, the new object will inherit the same value and will also be disabled.

Authorization requirements for the account used in the MCAUSER include:

- The ID should be in a group that has limited authority and is dedicated for use by MCAs.
- Authorization should be restricted to those queues to which the channel would legitimately be expected to place messages. For SVRCONN channels, the permissions granted should match those permissions that would be granted for a local (bindings mode) application. For channels between two queue managers, only grant access for the inbound channel to put and setall on the specific target queues needed, which may include the DLQ.
- Authorization to put to or get from SYSTEM queues should be restricted with limited exceptions. For example, desktop tools and instrumentation require access to put messages onto the system command queue but never to the system event queues.
- Inbound channels from adjacent queue managers should not be authorized to put messages directly onto transmission queues or the command queue unless this access is part of a deliberate design.

6.4.4 Avoid use of put authority context on channels

Setting an inbound channel to `PUTAUT(CTX)` causes the channel to put messages to target queues based on the authority of the user ID in the message header rather than the user ID that is associated with the channel connection. At first glance, this approach seems like the perfect solution to provide granular access control on a per-message basis and it does this function perfectly when the ID in the message is that of a low-privileged user. However, if the ID in the message is that of an administrator, the channel is not restricted as to where it will place the message. An attacker could use the channel to place messages on the command queue and execute arbitrary commands.

Setting `PUTAUT(CTX)` reduces administration by allowing several queue managers to share the same identity database, such as NIS+ or Windows Active Directory. The trade-off for that functionality is that it implicitly grants administrative rights to the adjacent queue manager. Therefore, if one queue manager is compromised, the attacker simultaneously gains access to all the adjacent queue managers. Although this functionality is often described as a security control, in fact it is best viewed as a compromise of security in return for convenience.

When there are few users and service accounts, the added risk outweighs any savings in administrative overhead. However, the risk of an attack rises with the population of users and accounts and at some point also outweighs the administrative benefit of context authority.

There may be an equilibrium point between these small and large populations of accounts in which a business case can be made to run with PUTAUT(CTX), but the nature of messaging systems is that they tend to grow over time. That equilibrium point may be only temporary and once context authority is used, disabling it can be difficult. Therefore, the authors do not see a valid case for using PUTAUT(CTX) in a secure environment.

6.4.5 Do not enable automatic channel definition

WebSphere MQ includes a feature to automatically define RECEIVER and SVRCONN type channels for incoming channel connections requests if the channel name does not already exist on the local queue manager. The channels are defined based on SYSTEM.AUTO.RECEIVER and SYSTEM.AUTO.SVRCONN. The functionality is enabled by the queue manager's Channel Auto Definition (CHAD) attribute as well as an exit point during the channel auto-definition process at which an exit program can intercept, modify, or reject the channel auto-definition attempt. Channel auto-definition is not available on the z/OS platform.

This capability provides an attack vector and, in the absence of a CHAD exit, allows uncontrolled creation of new channels. Fortunately, CHAD is disabled by default in a new queue manager. If an attacker gains administrative access, enabling channel auto-definition is one possible way to create a back door through which to gain access again at a later time.

A queue manager participating in a cluster will always auto-define cluster channels, regardless of the queue manager's CHAD setting. These channels also drive the CHAD exit point.

The queue manager's CHADEV attribute controls the generation of event messages for automatically defined channels. This attribute should be set to ENABLED so that an event message is generated whenever any channel is automatically defined. Cluster channels are created through the channel auto-definition mechanism and so will generate legitimate CHAD events, even if the queue manager's CHAD setting disables auto-definition of other channel types.

The queue manager's CHADEXIT attribute allows a custom-written exit module to influence the definition and initiation of automatically defined channels, including CLUSSDR channels. A CHAD exit can modify attributes on the channel, including the removal or addition of any supported channel exit module or exit data. If a CHAD exit is used, it should record full details about automatic definitions of channels to a log file, message on a queue, or a suitable monitoring or auditing facility.

6.4.6 Avoid using a default transmission queue

The name of the default transmission queue is defined by the queue manager's DEFXMITQ attribute. This attribute is set to blank when a queue manager is created so that there is no default transmission queue.

The default transmission queue is used when the queue manager needs to send a message to a remote queue manager, and both of the following conditions are true:

- The transmission queue name is not specified in a QREMOTE object.
- A transmission queue does not exist with the same name as the remote queue manager.

The result is that a message that does not resolve to any known destination will resolve to the default transmission queue and be forwarded to another queue manager. This feature can be used to an advantage in some WebSphere MQ designs, for example, hub and spoke topologies. But, it should not be utilized in all designs because an attacker can exploit this behavior to enumerate resources on the internal network or route messages to unauthorized destinations.

When the topology does not call for a default transmission queue, the WebSphere MQ administrator can use this behavior as an additional security control. WebSphere MQ does not require the default transmission queue to be served by a functional channel in order to put messages there. So, an administrator can define a queue, designate it as the queue manager's default transmission queue, and then monitor it. Any messages arriving on the queue indicate either a configuration error or an attack. This security control is commonly used on gateway queue managers that communicate with one or more third parties but can be useful on internal networks, as well.

The WebSphere MQ Telemetry components require a default transmission queue because each connected client resolves as though it were a queue manager. Although it would be possible to define a queue manager alias for each client, this does not scale and most implementations of WebSphere MQ Telemetry require a default transmission queue.

6.4.7 Avoid use of SERVER channels

SERVER channels are quite similar to SENDER channels. When they are started locally, they connect to a remote queue manager at the address defined by their `CONNAME` attribute, and then transport messages to that remote queue manager. The difference in the two channel types becomes apparent when they are started remotely by a REQUESTER channel.

When a SENDER channel is started by a requester, the inbound TCP socket is closed and then the SENDER channel initiates start-up as though it had been started locally. The SENDER channel then attempts to connect based on the value in the `CONNAME` attribute. However, when a connection request is handed to a SERVER channel, the network connection is used as is. The SERVER channel simply continues to negotiate the connection with the REQUESTER that contacted it. Therefore, the SENDER channel will only ever communicate with the IP address that is specified in its `CONNAME`, whereas the SERVER channel can potentially communicate with any IP address to which a network route can be established.

An attacker can exploit this behavior by crafting a REQUESTER channel that is designed to connect to the SERVER channel at the target queue manager. This REQUESTER channel can be used to siphon off messages that are destined for a legitimate destination that is normally served by the hijacked channel. If the attacker can also successfully deliver messages to the target queue manager on a return channel, a full request/reply exchange is possible.

It is a common misconception that REQUESTER channels only work with SERVER channels, but this is not true. Since a REQUESTER can start SENDER channels as well as SERVER channels, it is best to use a REQUESTER/SENDER pair when there is a requirement that the connection must be startable from the receiving side. This scenario is common in B2B scenarios and any time that a firewall separates two queue managers.

A REQUESTER/SERVER pair is called for only when there is a legitimate requirement that the SERVER channel must connect to multiple possible destinations without manual intervention to update the `CONNAME` value. Traditionally, this requirement has been associated with disaster recovery scenarios but multi-instance `CONNAME` functionality has eliminated that requirement. As a result, there are extremely few legitimate use cases for SERVER channels in modern queue managers.

If SERVER channels are used, mutually authenticated TLS or SSL with appropriate SSLPEER or CHLAUTH rules should be used to authenticate the connection.

6.4.8 Restrict access to transmission queues

When an application opens a queue, the queue manager performs a name resolution process to see if the destination is a local or remote queue. If a remote destination is found, the message is put to a transmission queue serving a channel that leads to the remote queue manager. The local queue manager will automatically add a correctly formatted Transmission Queue Header (MQXQH) to the message data. Alternatively, an application program can open the transmission queue directly. In this case, it will need to add a correctly formatted MQXQH.

Because the name resolution process stops once the transmission queue has been identified, an application that is authorized to put messages directly to a transmission queue can address those messages to any queue on the target queue manager. Although it is the receiving queue manager's responsibility to set the inbound channel's MCAUSER and restrict the queues to which the channel can put messages, restricting destination addresses on the sending queue manager provides an additional control for administrators wanting to implement a *defense-in-depth* approach.

Rather than granting access directly to transmission queues, administrators can define QREMOTE objects or QALIAS objects locally that refer to the remote destinations. Authority profiles can then be used to grant the ability to put messages to these objects while restricting access to the transmission queues.

As of WebSphere MQ version 7.1, authorizations can now be granted to remote queue manager names without granting access to the transmission queue directly. This capability is useful in a clustered network where it provides a level of granularity between per-queue authorization and per-cluster authorization. However, in any topology, it is equivalent to granting access to a transmit queue in that the control does not provide per-queue authorization granularity for remote destinations.

From a policy standpoint, the receiving queue manager should not rely on the sending queue manager to properly restrict access to its transmission queue. The primary responsibility for restricting destinations of inbound channels between queue managers remains with the receiving queue manager. Setting a low-privileged MCAUSER on the receiving channel and restricting its access should be considered mandatory. Restricting access to transmission queues on the sending side is a good practice but is always considered a secondary security control.

6.4.9 Increase message retry on channels

When a Message Channel Agent (MCA) cannot put an incoming message to the destination queue and a message-retry exit is not defined, it will periodically retry the message at set intervals, for a limited number of tries. After exhausting the retry count, the message will be put to the DLQ.

The channel attributes `MRTMR` and `MRRTY` control the retry interval time and the count of attempts. The default settings are 1,000 milliseconds (1 second) and 10 attempts. With these settings, about 3,600 messages can be put to the DLQ per hour by each channel instance.

Increasing the values of these attributes will reduce the rate at which the DLQ can be hit by an attacker or rogue application. This action reduces the exposure to a resource exhaustion attack that gradually consumes all disk space that is available for queues.

For example, the attribute values can be increased by a factor of 10 by using the following MQSC command:

```
ALTER CHANNEL(....) CHLTYPE(....) MRTMR(10000) MRRTY(100)
```

The result is that each retry cycle is 1,000 seconds, instead of 10 seconds. One message could be put to the DLQ for each retry cycle. So, this configuration will allow only 36 messages per hour to be put to the DLQ by running instances of this channel. The values should be tuned based on security requirements and experience with message retry occurring in your organization.

6.4.10 Use the managed listener

Early versions of WebSphere MQ on UNIX relied on the system network daemon process (**`inetd`**) to listen for channel connection requests and then start a new Message Channel Agent process (**`amqcrsta`**) for each incoming channel connection request. Because the system limits the number of processes that an ID can run simultaneously and because the number of TCP ephemeral ports is finite, this approach is vulnerable to a Denial of Service (DoS) attack on both system and WebSphere MQ resources. It also requires system administrator (root) privileges to initially configure and maintain the `/etc/services` and `/etc/inetd.conf` files.

Although all modern versions of WebSphere MQ provide a managed listener process that is highly scalable, migration to newer versions of WebSphere MQ may not have included migration to that component and away from `inetd`.

Ensure that the following conditions are met:

- WebSphere MQ ports are not defined in the `/etc/services` file or the `/etc/inetd.conf` file on UNIX platforms.
- **`runmqlsr`** is started or stopped by the queue manager rather than by external scripts or process management tools.
- Queue managers have one or more LISTENER objects that are configured to use the correct port and protocol, and are managed by the queue manager start-up and shutdown, for example:

  ```
  DEFINE LISTENER('TCP.1414') TRPTYPE(TCP) PORT(1414) CONTROL(QMGR)
  ```

- Queue managers have a listener dedicated for administrative access.

These actions provide much better scalability for listeners, direct control by the WebSphere MQ administrator, and the ability to respond to security incidents swiftly and effectively.

6.4.11 Specify local address on outbound channels

The LOCALADDR attribute of several types of WebSphere MQ channels can be used to bind an outbound channel to a specific local IP address. For example, a multi-homed server may have multiple physical IP addresses and one or more virtual IP addresses. If the channel's LOCALADDR attribute is left blank, the queue manager allows the TCP/IP stack to select the best route. When multiple addresses are eligible, the queue manager does not know in advance which address will be used. By specifying LOCALADDR, the queue manager can force the connection to bind to a specific address among those addresses that are eligible.

This attribute has a number of uses from a security perspective. One of the most common is that of a hardware cluster where the queue manager has a physical IP address and a virtual IP address provided by the cluster. Ideally, the firewall or remote queue manager should see connections from the local queue manager originating from the virtual IP address rather than one of the physical IP addresses. Unfortunately, the physical IP address is usually weighted higher in the routing table and binds the outbound connection. The LOCALADDR attribute is used to make the outbound channel bind to the virtual IP address to eliminate this problem.

Another use of LOCALADDR is on a B2B gateway queue manager. Typically, this queue manager has two IP addresses where one faces the external network and one faces the internal network. The administrator can use LOCALADDR on the internal-facing channels to prevent them from being used to communicate with the external network. Similarly, LOCALADDR can bind the external facing channels to the external facing IP address and prevent them from connecting to the internal network.

In high security configurations, the queue manager will have a dedicated IP address on a separate subnet in order to segregate administrative traffic from application traffic. In this case, the LOCALADDR attribute can be used as an additional control to enforce that segregation of traffic.

6.4.12 Usage of port numbers

A queue manager can listen on many TCP/IP ports. The port numbers do not need to use the default values that often appear in the WebSphere MQ Information Center or other public sources, such as 1414, 1415, and 1516. As a security control, many administrators assign WebSphere MQ listeners to non-standard ports. As a general rule, avoiding well-known names and changing vendor defaults are good security practices. However, remember that TCP/IP ports are finite and the entire range can be scanned in a matter of seconds. Therefore, the use of non-standard ports is an effective defense against casual or accidental attacks but not against a skilled attacker.

Another case where multiple listeners are useful is to provide isolation or classes of service. It is a common practice on a gateway queue manager to assign a dedicated listener and port for each external B2B partner, another dedicated listener and port for internal application traffic, and another dedicated listener and port for administrative traffic. This approach allows selective shutdown of any B2B partner or application listener, or even complete shutdown of all listeners except the administrative listener.

Automation tools can be used to start and stop listeners on channels based on when they need to be available for legitimate use. Firewalls can be configured to restrict the usage of port numbers by remote IP addresses.

6.4.13 Queue manager to queue manager versus clients

Queue managers can accept connections from other queue managers or from clients. When considering the topology of the messaging network, remember the security aspects of the different types of connections.

When a connection originates from another queue manager, the code that drives API calls against the local queue manager is the message channel agent. This code is vendor code that is written by IBM, operates under control of the local administration team, and will only ever execute `CONNECT`, `INQUIRE`, `OPEN`, and `PUT` calls. This design limits an attacker's ability to access operating system and queue manager resources. However, local administrators have no control over the user ID and other context information carried by the messages that arrive, unless they implement a message exit that is capable of enforcing policies on these fields.

By contract, when connections originate from client applications, the code that drives API calls on the queue manager is the client application itself. For connections that originate from external parties or desktop tools, the administrator has no guarantee of the provenance of the code making the API calls or what behavior it might exhibit. For example, the application might attempt to enumerate queue names by cycling through likely values. If it receives a reason code of `2085 MQRC_UNKNOWN_OBJECT`, the queue does not exist. A return code of `2035 MQRC_AUTHORIZATION_ERROR` informs the attacker that the object does exist. However, with client connections, the WebSphere MQ administrator has the option to strongly authenticate the connection and can thus implement strict controls over the identity and other context information carried by messages in the network.

These differences tend to be most relevant in B2B situations where the remote entity is by definition untrusted and outside the control of the local administrative team. In this case, the ability of the client to execute arbitrary API calls tends to outweigh the advantages, and so traditionally, B2B interfaces have used queue manager to queue manager connections rather than clients. To address the issues with message identity context, a message exit has been provided in the appendix of this book and is featured in the B2B scenario.

6.4.14 Separate channels for application messaging

Multiple channel paths can be implemented between queue managers to allow message flows to be separated for multiple applications and for different types of messages within an application, for example, small and large, or high priority and low priority. This implementation also allows control of message flow at the channel level, such as stopping a channel if there is an abnormal flow of messages due to application behavior, or an attacker attempting to penetrate applications or use DoS methods.

6.5 Queues and other objects

This section describes security controls for queues and other objects that can be used by applications. The objective is to prevent unauthorized access and operations that could impact the proper operation of the queue manager.

6.5.1 Restrict access to system default objects

The only objects that are likely to exist on the majority of queue managers are those objects that are created by default with a new queue manager. Because the default object names are well known and almost certain to exist, writers of documentation and tutorials commonly refer to these objects. For example, `SYSTEM.DEFAULT.SVRCONN` and `SYSTEM.DEFAULT.LOCAL.QUEUE` are both frequently used as the basis for Hello World program examples.

A side effect of this long-standing practice is that many administrators routinely grant access to system default objects to non-administrative users. This policy creates a dependency that these objects must remain fully functional and therefore prevents a security-conscious administrator from disabling them.

For example, the defaults for `SYSTEM.DEFAULT.LOCAL.QUEUE` include `MAXDEPTH(5000)` and `MAXMSGL(4194304)` and these represent 2 GB of disk storage per queue. Any new queues that are defined that inherit these values also potentially can consume 2 GB of space before they fill. As a security control, the MQ administrator may want to set lower values on these default objects but the necessity of allowing their access by random users conflicts with that goal. The simplest and most effective solution is to reserve all `SYSTEM.DEFAULT.*` objects and define new objects for non-administrative applications and users.

6.5.2 Least access authorization model

One of the fundamental principles of security design is to limit authorizations to the absolute minimum that is required. For WebSphere MQ, implementing a least access model requires the administrator to understand some of the subtleties of the queue manager's behavior in order to avoid unintended overauthorization. In particular, permissions are best granted to groups rather than principals, and the administrator must remember that some specific permission settings have impacts beyond the objects to which they primarily apply.

WebSphere MQ is designed to grant access to groups rather than principals. For most platforms, issuing a command to grant access to a principal causes the queue manager to perform a lookup against that principal to determine its primary group. The access control list entry is then entered against that group rather than the principal that the administrator specified. This approach may result in unintended behavior, depending on which group is primary for the account that is specified. If the primary group is actually `staff` or `users`, the administrator has just authorized a very broad group, likely one that includes all users of the system.

The exception is the Windows platform, which supports authority profiles for principals (user IDs). Role-based authorization is easier to manage if it is based on groups, so authority to principals should not be used.

The following OAM authorities should not be granted for security profiles on objects that are used by applications:

- `+allmqi`, `+alladm`, and `+all` are generic authorities that group individual authorities, such as `+inq`, `+browse`, `+put`, `+get`, and others. All of the generic authority specifiers contain privileged authorities that are not generally required in application designs, such as alternate user authority `+altusr`, context authorities `+passid`, `+passall`, `+setid`, and `+setall`, and administrative authorities `+chg` and `+dlt`.
- `+crt` authority on any object will result in the group being able to create new objects of that type with any name. Granting `+crt` for queues indirectly grants full administrative privileges to the recipient.

- `+chg` or `+dlt` on an object will allow attributes to be changed or the object to be deleted. This function is an administrative function that an application would not normally perform.
- `+set` on the queue manager object provides wide ranging administrative authorities to the queue manager. In addition, granting `+set` indirectly grants full administrative access to the recipient.

Administrative access authorities are also discussed in 4.10, “Authorizations that grant administrative access” on page 28.

6.5.3 Authority to SYSTEM.BASE.TOPIC

Application groups should not have `+pub`, `+sub`, or `+resume` authority to the `SYSTEM.BASE.TOPIC` object. This access would allow the group to publish, subscribe, or resume a subscription to any topic in the topic tree. Instead, select a level within the topic tree below the root and apply authorizations uniformly across that level. When choosing the depth, select a level far enough down the topic tree to provide sufficient authorization granularity but not so far that administration becomes too granular and thus burdensome.

6.5.4 Considerations for dead letter queue and topics

In WebSphere MQ version 7.1 and later, the USEDLQ attribute can be set on TOPIC objects to determine the action for publication messages that cannot be delivered to the correct subscriber. The default value for the USEDLQ attribute of `SYSTEM.BASE.TOPIC` is `YES` so that these messages are sent to the queue manager’s DLQ. Setting USEDLQ to `NO` causes the queue manager to treat messages that cannot be delivered to all subscribers as a failure to put the publication message and no record will be made. The application attempting to publish the message will receive an exception and should take appropriate corrective action.

Setting USEDLQ to `YES` provides visibility to messages that are undeliverable for a variety of reasons, including authorization errors. However, allowing publications to land in the DLQ creates an opportunity for a resource exhaustion attack. An application that floods the DLQ may be able to consume all available space in the file system hosting the queues. The choice of which setting to use should consider these factors.

6.6 Applications using WebSphere MQ

WebSphere MQ provides features that can be very useful and convenient to use in application designs, and these features can impact the security model. This section describes some of these potential impacts.

6.6.1 Avoid setting message context fields

There are eight context fields in the MQ message descriptor (MQMD) structure that are populated by the queue manager and returned to the application after it puts a message to a queue:

- UserIdentifier
- Accounting Token
- ApplIdentityData
- ApplOriginData
- PutApplType
- PutApplName

- PutDate
- PutTime

The fields can also be inspected by the consuming application in order to find out information about the originator.

Application groups can be granted `+passid`, `+passall`, `+setid`, or `+setall` permissions on the authorization profiles that control access to queues. These permissions allow the context fields to be set by the application program, which puts a message, overriding the queue manager defaults.

The intention of this feature is to allow a trusted application to propagate the message context information to a downstream application. For example, a routing application may inspect a region code before forwarding a message on to one of several back-end applications for processing. The application would get the original message under syncpoint, determine the destination, put a copy of the same message on the appropriate queue, and then commit the transaction. By default, the copy that is received by the back-end system would have the context information of the routing application rather than the original sender. But by granting the routing application the appropriate permissions, it is able to pass the context from the original message on in the copy.

Another example is that of a web service that resolves the identity of the requestor and must pass that identity on to an application in a request message. Since there is no existing message from which to pass the identity context, the web application must explicitly set the identity context of the messages that it sends.

Both of these examples allow propagation of message identity through an intermediate system. The security implication is that the authorization group to which the intermediate application belongs has the ability to spoof identities of messages sent to the back-end systems.

In general, applications should not be designed to pass or set message context fields. Applications that legitimately require these capabilities should be subjected to an elevated level of inspection and stricter change control gateways in order to obtain such trusted status. Context fields are described in more detail in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fg10860_.htm

6.6.2 Avoid alternate user ID

Normally, WebSphere MQ performs authorization checks against the process user ID of an application program. A suitably authorized application can nominate an alternate user ID for the authorization check. This feature requires the application group to have `+altusr` authority on the authorization profile that controls access to the object.

This technique can be useful as a means of identity propagation where one application acts on behalf of another application or a remote user. For example, a web application accepts service requests from users and fulfills them by using a request/reply message exchange. Rather than opening the request queue under its own ID, the web application nominates the ID of the logged-in user. If the user is authorized to the request queue, the open succeeds and the transaction continues. Otherwise, the web application receives an authorization error and reports this failure to the logged-in user.

A security concern is that the program can assert any alternate user ID, including user IDs that are WebSphere MQ administrators and thus guaranteed to succeed. For this reason, alternate user ID authorization should not be granted routinely. For those programs that legitimately require `+altusr` authority, grant the privilege only on those queues that require it, document the requirement, and periodically reconcile the actual permissions that are granted to those that are documented.

6.6.3 User ID and password fields on client connections

WebSphere MQ provides the means for a client application to pass user ID and password credentials when requesting connection to a queue manager. These fields are made available to security exit programs that can validate them as required, or passed to the OAM. However, no password validation exit or other native feature in WebSphere MQ version 7.5 or earlier versions performs user ID and password credential validation. In the absence of an exit, WebSphere MQ accepts the ID that is presented and ignores the password.

It is possible to implement exits to provide strong authentication against a trusted security service, but unless your organization has done so, it is best to avoid sending passwords over client channels. In those cases where a security exit that validates user ID and passwords has been implemented, be sure that the credentials are transmitted by using session encryption. The easiest way to insure that credentials are not exposed is to configure the channel to use a TLS cipher suite with strong encryption.

The queue manager configuration dialog in recent versions of WebSphere MQ Explorer includes fields to specify the user ID and password that are used for the connection request. The Java and Java Message Service (JMS) classes include the fields in their connect methods. Since the application or user can trivially set these fields, do not rely on them. Instead, set the channel's MCAUSER to a non-administrative value that overrides whatever value is presented by a client, use a channel exit to set the MCAUSER, or in recent versions of WebSphere MQ, use CHLAUTH rules to set the MCAUSER.

6.6.4 Use segregated input queues

Rather than providing a single queue to handle all incoming messages to an application, it is more secure to use a design where each service or function of an application has its own input queue. Put authority to the queue can be granted only to the application groups that need to produce messages. Get authority to the queue can be granted only to the application group that consumes messages. Attack or misuse of one queue or application function can be insulated from impacting other application functions. If an incident occurs, the queue can be put disabled or get disabled to prevent the flow of rogue messages. The same principles can be used to provide application function-based authorization to reply queues and other queues that implement messaging interfaces between applications.

The value of segregating input queues begins to diminish as the number of queues grows larger. At some point, the complexity of managing great numbers of queues exceeds the benefit. The break-even point depends on the configuration management tools and processes that are used by the administrators.

6.6.5 Careful use of report messages

WebSphere MQ provides the ability for an application sending a message to request reports that are based on the eventual delivery of the message or any exceptions encountered in the delivery of the message. These report messages can optionally contain the original message in its entirety. The most common problem with report messages is that a legitimate and well-intentioned application can inadvertently execute a resource exhaustion attack by doubling or tripling network traffic or flooding a DLQ with large report messages.

For example, consider an existing application that is changed to request Confirmation on Arrival (COA) and Confirmation on Delivery (COD) reports. The result is that the network must now transport the original message and both confirmation report messages, potentially tripling traffic on the network. If the network was already operating at 50% of capacity or higher, this change will result in an outage.

Another potential issue when report messages are not consumed promptly or routed correctly is that they can accumulate in their destination queue or the DLQ and consume all available space in the file system that hosts the queues.

Although it is possible that an attacker might attempt to exploit these features, no such attack has been publicly reported to date. However, it is somewhat common that legitimate applications unintentionally cause some of these impacts through errors in coding or design. When you want applications to use the report message capabilities of WebSphere MQ, these capabilities should be well tested prior to implementation in production. Consideration should be given to whether the features should be enabled full time or selectively enabled for problem determination. As a rule, use the smallest report message that provides the required functionality.

For further details about report messages, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fr13320_.htm

6.7 Recent changes

The items in this section cut across many of the other categories but are grouped together for the benefit of those readers that are familiar with WebSphere MQ v7.0 and earlier. These represent security-related features that are new or updated as of WebSphere MQ v7.1 or related to the move from Global Secure ToolKit (GSKit) 7 to GSKit 8.

For further information about new features and changes, see the WebSphere MQ Information Centers:

- http://publib.boulder.ibm.com/infocenter/wmqv7/v7r1/topic/com.ibm.mq.doc/mq50090_.htm
- http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/mq50095_.htm

6.7.1 Dedicated cluster transmission queues

In WebSphere MQ version 7.5 for distributed platforms, it is possible to use dedicated transmission queues for cluster sender channels, rather than all clusters sharing use of `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. This capability provides separation of traffic on clusters and reduces exposure to messaging attacks and their effects. This feature is new in WebSphere MQ version 7.5.

6.7.2 WebSphere Message Broker default configuration wizard

The WebSphere Message Broker default configuration wizard (DCW) will create a new queue manager if one does not already exist. For versions of WebSphere MQ that support CHLAUTH records, the DCW will disable CHLAUTH records for any queue managers that it creates.

In all cases, whether the queue manager exists or the DCW creates a new one, the DCW creates a `SYSTEM.BRK.CONFIG` channel but does not configure any security controls on it. On a queue manager with no CHLAUTH rules, this design results in a channel that allows anonymous remote code execution on the queue manager's host and control of the MQ administrative account by anyone with an IP route to the queue manager.

To address this exposure, CHLAUTH should be reenabled and appropriate CHLAUTH rules, MCAUSER setting, and TLS/SSL protection should be put in place to ensure that the channel is strongly authenticated and appropriately authorized.

6.7.3 MCA interception for clients

MCA interception for clients is a new feature in WebSphere MQ version 7.5 that allows clients outside of a WebSphere MQ AMS queue manager to encrypt and decrypt messages. The client should also be strongly authenticated and the channel traffic encrypted, using TLS/SSL and channel authentication (CHLAUTH) rules.

6.7.4 RFC 5280 certificate validation policy

A new feature in WebSphere MQ version 7.5 allows specification of how strictly the certificate chain validation conforms to the RFC 5280 industry security standard.

6.7.5 FIPS compliance on SSL/TLS and AMS

Federal Information Processing Standard (FIPS) compliance requirements are complex and may change over time and how they are implemented in WebSphere MQ. As a starting point, see the WebSphere MQ Information Center:

`http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy11005_.htm`

If FIPS-certified cryptographic hardware is present on your system, WebSphere MQ can be configured to use the modules provided by the hardware manufacturer rather than use its own cryptography package, IBM Crypto for C (ICC).

6.7.6 New CipherSpecs and CipherSuites

WebSphere MQ version 7.1 introduced support for additional digital certificate signature and cipher algorithms to provide compliance with FIPS 140-2. There are no additions in version 7.5. The algorithms are also available in GSKIT version 8, which is supported by WebSphere MQ version 7.0.

6.7.7 NSA Suite B support

WebSphere MQ version 7.1 and 7.5 can be configured to conform to the Suite B-compliant profile of TLS version 1.2, as recommended by the US National Security Agency (NSA).

WebSphere MQ does not support the Suite B transitional protocol. For further details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy11025_.htm

6.7.8 Distinguished Encoding Rules in SSLPEER and SSLCERTI

WebSphere MQ version 7.1 and later implements the Distinguished Encoding Rules (DER) encoding of a certificate and uses that standard to determine the subject and issuer distinguished names. These names appear in the `SSLPEER` and `SSLCERTI` fields. The `SERIALNUMBER` attribute is also included in the subject distinguished name and contains the serial number for the certificate of the remote partner.

6.8 Procedural considerations

This section describes some of the human processes that contribute to overall security of the messaging network and surrounding systems.

6.8.1 Software currency

IBM regularly releases fix pack software updates for WebSphere MQ by using the Authorized Program Analysis Report (APAR) and Program Temporary Fix (PTF) methodology. Occasionally, these updates address security-related issues. Full details are not usually disclosed, other than stating that a security vulnerability has been identified and remediated in an area of the product. It is a good practice to review all fix packs when they are released and consider installation in your environments, which will reduce the exposure of your organization to known security issues in WebSphere MQ. Preventive service planning information for all WebSphere MQ versions and platforms is available on Internet:

http://www.ibm.com/support/docview.wss?uid=swg21254675

6.8.2 Periodic revalidation of security roles

Your organization should periodically review all user IDs and groups on the system that have WebSphere MQ related security roles to ensure that they are currently valid. This revalidation applies to administrator roles and to application roles for various authorities to use objects.

People may have left the organization or moved to a different role. Applications can be redesigned or decommissioned. Security administrators may have accidentally or inappropriately added a user ID to a group that has a security role in WebSphere MQ. In many cases, the WebSphere MQ administrator will not be aware of what people and application user IDs should be in what groups and will need to contact the owner of the user IDs or groups to confirm the revalidation of their roles.

If exceptions are found, it may be necessary to remove user IDs from groups or modify WebSphere MQ authority profiles. This process should be done in agreement with your organization security policies and standards for application design.

6.8.3 Resource monitoring to detect and record security incidents

Many WebSphere MQ queue manager implementations include external monitoring and alerting of resource utilization, such as disk file system usage, CPU usage, and network usage. WebSphere MQ infrastructure and object monitoring is also performed, such as queue depth events, queue manager processes, channel status, error logs, and event messages. This monitoring can also be used as a tool to detect and record security incidents, such as excessive resource usage by DoS attacks, authorization failures, and unusual patterns of channel usage and messaging.

Monitoring for security purposes can also be extended to detect and audit authorized and unauthorized changes to WebSphere infrastructure, including the following information:

- Object attributes
- Status of channels
- Static messages on system queues and application queues
- Attributes in queue manager configuration files
- Key repository files
- Authorization profiles
- Group memberships
- User ID attributes
- Permissions on directories and files
- Usage of administrator user IDs (for example, `sudo` on UNIX)
- Usage of administrator channels and interfaces
- Keystroke logging

7

Operating system specifics

The scenarios and techniques that are presented in this book were selected for their applicability across the most widely used use cases and broadest platform coverage. However, there are sufficient variations across the platforms that are supported by IBM WebSphere MQ to warrant a chapter that is dedicated to those differences.

This chapter contains important information and considerations regarding WebSphere MQ security for z/OS, IBM i, and Microsoft Windows operating system platforms. This information is not intended to be a complete list of differences or comparisons of security with other platforms.

This chapter includes the following topics:

- IBM z/OS
- IBM i
- Microsoft Windows

7.1 IBM z/OS

The fundamental features of WebSphere MQ on the z/OS operating system are directly comparable with other platforms. Programs that use the standard Message Queue Interface (MQI) calls can be ported to z/OS. WebSphere MQ channels can interoperate with all supported queue manager and client platforms. There are however a large number of additional features and implementation-specific differences for installing, configuring, programming, and operating WebSphere MQ on z/OS. This section outlines the key differences for security. Links to the WebSphere MQ Information Centers are provided for the most comprehensive and current information.

7.1.1 WebSphere MQ security management

WebSphere MQ uses System Authorization Facility (SAF) for authorization checking on the z/OS platform. SAF invokes an External Security Manager (ESM), such as RACF, TopSecret, or ACF2, to service security requests. This approach is very different from the object authority manager (OAM) that is used on distributed platforms.

On z/OS, additional authorization controls are available on command security, categories of security checks can be switched on or off, and object security is implemented in a different manner. Additional auditing and diagnostic features are provided by SAF. Granular security controls are implemented for the WebSphere MQ channel initiator, clusters, queue sharing groups (QSG), CICS, and IMS interfaces.

For complete information about configuring and managing security and for special considerations for security on the z/OS platform, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs12080_.htm

7.1.2 TLS/SSL certificate and key repository management

WebSphere MQ on z/OS uses SAF and z/OS System Secure Sockets Layer (SSL) for management of Transport Layer Security (TLS)/SSL, digital certificates, and key repositories. There are many OS-specific concepts and tasks for setting up and working with these facilities that are completely different from any other platform. A full description of all the differences is beyond the scope of this book. For details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy12440_.htm

For the samples that are used for this book, see “Managing certificates on z/OS” on page 308.

7.1.3 Queue sharing groups

The Queue Sharing Groups (QSG) feature of WebSphere MQ for z/OS allows multiple queue managers running on different logical partitions (LPARs) to share access to a group of local queues. Security checks occur at the QSG level and the queue manager level. There are also switch profiles to control which checks are performed. For details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs12170_.htm

7.1.4 Channel types have additional values of PUTAUT

Receiver, requestor, and cluster receiver channel types have additional values of the PUTAUT attribute on z/OS. This attribute determines the source of user ID that a Message Channel Agent (MCA) will use for alternate authority to put messages to destination queues. It can also be used on server connection channels on the z/OS platform to determine the user ID for MQI calls.

On distributed platforms, the attribute can take the values Default Security (DEF) or Context Security (CTX). Two additional values are available on z/OS: Only MCA Security (ONLYMCA) and Alternate MCA Security (ALTMCA). These values are described in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/ic11930_.htm

7.1.5 Separating put and get authority

The WebSphere MQ usage of SAF on z/OS does not allow `put` and `get` authority to be individually controlled on security profiles for queue objects. There is an indirect method to distinguish these authorities by using security profiles on two alias objects. For details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs12290_.htm

7.1.6 Publish/subscribe security

The introduction of the internal publish/subscribe engine in WebSphere MQ for z/OS version 7.0 also added support for authorization profiles that more effectively protect objects that have mixed case names. Previously, profile names were restricted to uppercase only names. New SAF classes were provided to support mixed case: MXADMIN, MXPROC, MXNLIST, MXQUEUE, and MXTOPIC.

Part of the migration to WebSphere MQ for z/OS version 7.0 from any previous versions involved an optional step, “Migrating a queue manager to mixed case security”, which explains how to activate the new MX classes.

If this step is not completed and you have queue managers running WebSphere MQ version 7.0 or later using the old MQ classes, and publish/subscribe is enabled and being used, there are no authorization controls applied. Even if the profile `qmgr.SYSTEM.BASE.TOPIC` has been defined in class MXTOPIC with all access removed, it is not actually being checked and no diagnostic messages are produced on the system log.

The preferred practice is to migrate queue managers to use MX classes. If that is not possible, turn off publish/subscribe by setting the queue manager attribute `PSMODE(DISABLED)`.

For a basic Security Check sample program, see “Simple z/OS MQ security check” on page 314.

Migration is described in the WebSphere MQ Information Center:

- http://publib.boulder.ibm.com/infocenter/wmqv7/v7r1/topic/com.ibm.mq.doc/mi21470_.htm
- http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/mi21480_.htm

7.1.7 Certificate sharing in a queue sharing group

Queue managers on z/OS can share a private certificate in a QSG. There are two methods to share a private certificate in a QSG:

- The Channel Initiator (CHIN) address space of all queue managers is running under same user ID, and set the queue managers' SSLKEYR attribute:

  ```
  ALTER QMGR SSLKEYR(CSQ1RING)
  ```

- Specify the owner of the key ring in the queue managers' SSLKEYR attribute:

  ```
  ALTER QMGR SSLKEYR('userid/CSQ1RING')
  ```

  The "guest" queue managers need `ACC(UPDATE)` authority to the `IRR.DIGTCERT.LISTRING CL(FACILITY)` resource in SAF. Normally, ACC(READ) is sufficient when the user owns the key ring.

On z/OS, the queue manager's certificate label must be in the format `ibmWebSphereMQ`*`QsgName`*, for example, `ibmWebSphereMQQSG1`, where `QSG1` is the name of the QSG.

For further details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs12870_.htm

7.1.8 RESLEVEL security

RESLEVEL is a very powerful option; it can cause the bypassing of all resource security checks for a particular connection.

If you do not have a RESLEVEL profile defined, you must be careful that no other profile in the MQADMIN class matches hlq.RESLEVEL. For example, if you have a profile in MQADMIN called hlq.** and no hlq.RESLEVEL profile, beware of the consequences of the hlq.** profile because it is used for the RESLEVEL check.

Define an hlq.RESLEVEL profile and set the UACC to `NONE`, rather than have no RESLEVEL profile at all. Have as few users or groups in the access list as possible.

> **Important:** If a user is granted either CONTROL or ALTER access to the RESLEVEL profile, it means that security checks for that user are bypassed.

For further details, see the WebSphere MQ Information Center:

http://publib.boulder.ibm.com/infocenter/wmqv7/v7r1/topic/com.ibm.mq.doc/zs12440_.htm

7.2 IBM i

WebSphere MQ on the IBM i operating system is very similar in basic operation and command structure to the UNIX and Windows platforms, including the use of OAM as the WebSphere MQ security manager. However, the user operational interface, command names, and command arguments are markedly different. There are also significant differences for security, some of which are briefly described in this section.

7.2.1 Special users

The WebSphere MQ installation creates two special profiles in IBM i security. The QMQM user profile is used for internal product functions. The QMQMADM group profile is used for WebSphere MQ administrators. For details about setting up security for IBM i, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/ia11280_.htm

7.2.2 Command authorization

By default, authority to issue WebSphere MQ control commands is restricted to members of the QMQMADM group. All other users are specifically excluded. Setting up IBM i security to allow other users to issue commands is described in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/ia113200_.htm

The requirements for administering WebSphere MQ are also described:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy10760_.htm

Authority to issue commands should be limited to the user's role as WebSphere MQ administrator and based on your organization's security requirements.

7.2.3 Key repository

The key repository consists of a certificate store and password stash file. They can be owned by the queue manager or managed by the system by setting the queue manager attribute SSLKEYR(*SYSTEM). For further details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy10970_.htm

7.3 Microsoft Windows

The Microsoft Windows implementation of WebSphere MQ is very similar to UNIX platforms, including the use of OAM as the security manager. This section describes a number of important differences and examples of special considerations for security on this platform. For further details about setting up UNIX and Windows security, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa12730_.htm

Security topics for Windows are also described at this website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa13360_.htm

7.3.1 Specific profiles for principals and groups in OAM

On Windows, access control lists (ACLs) in OAM can be based on principals (user IDs) or groups. OAM stores the Windows Security Identifier (SID) that uniquely identifies a user ID or group, rather than its name. A SID is a binary number that is often displayed as a long string of hexadecimal digits and hyphens.

User ID and group names can be qualified with an Active Directory domain name, which should be used wherever possible to avoid any confusion about the domain in which the

name exists. Additional details about this issue are provided in 4.11.2, “Granting access to principals” on page 30.

The differences between the Windows and UNIX treatment of principals and groups are described in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa12800_.htm

Windows also allows groups to be nested within groups, although some restrictions apply when using these nested groups with WebSphere MQ. For more information, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa13420_.htm

7.3.2 Deleting user IDs or groups

When a user ID or group is deleted on Windows, its SID becomes invalid and it can never be reused. WebSphere MQ continues to store the deleted SID in OAM authorization profiles and this SID cannot be deleted from OAM by using the **setmqaut** command. To avoid this situation, delete all the OAM profiles that refer to a principal (user ID) or group before deleting the user ID or group from Windows.

If you have OAM profiles for deleted SIDs, the only supported method to remove these redundant profiles is to follow these steps:

1. Back up the configuration of the queue manager, including OAM authorization records, by using the dmqmqcfg program in WebSphere version 7.1 or later, or SupportPac MS03 for older versions.
2. Record the queue manager initialization settings, such as log file type, numbers, and sizes. Back up other queue manager customizations and exits if they are present. Unload messages from application queues if they need to be preserved.
3. Delete the offending authorization records from the backup file.
4. Shut down and delete the queue manager.
5. Re-create the queue manager with the same log file type, numbers, sizes, and customizations.
6. Apply the edited backup of OAM authorization records. Apply the object definitions.
7. Load any application messages to queues, as required.

The Queue Manager Identity (QMID) changes when a queue manager is re-created with the same name. This change will have ramifications if the queue manager is a member of WebSphere MQ clusters.

7.3.3 Service user ID using active directory

The user ID of the Windows service for WebSphere MQ needs to be able to query the group membership of domain user IDs in Active Directory. It may be necessary to change the default user ID MUSR_MQADMIN to another user ID that has the necessary rights. For further details, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa12210_.htm

7.3.4 Application event log

WebSphere MQ writes information and error messages about its operation to a number of its own error log files. This information includes authorization failures and other security-related messages. On the Windows platform, WebSphere MQ also writes these messages to the Windows application event log. Use the event log viewer program eventvwr.exe that is supplied with Windows to view the application event log. The log can be saved to a file as tab-delimited or comma-delimited records for further analysis.

7.3.5 Securing shared data for multiple instance queue managers on Windows

Local or global alternative security groups can be used to secure shared data folders that are used by multiple instance queue managers. For further details, see the WebSphere MQ Information Center:

- http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa71020_.htm
- http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa71090_.htm

8

Scenario preparation

The scenarios that are presented in the remainder of this book represent a diverse range of use cases and examples of business solutions. They demonstrate the implementation of IBM WebSphere MQ security using product features and design techniques.

For simplicity, they all use common infrastructure environments on UNIX and Microsoft Windows platforms. No particular software products or solutions, industry segments, or business operating models are targeted.

This chapter includes the following topics:

- Overview
- Servers and network topology
- Operating systems and infrastructure software
- Operating system configuration
- WebSphere MQ installation and configuration
- Other software installation and configuration
- Naming standards and conventions
- Certificate authorities
- OCSP responder
- LDAP server to host CRLs
- WebSphere MQ (CMS) keystores
- Other certificate tools

8.1 Overview

The scenarios are designed to demonstrate aspects of WebSphere MQ authentication and authorization capability. Every scenario requires some basic capability within the infrastructure. Environment setup, installation, and configuration tasks must be performed before attempting any of the scenarios.

In order to present certificate revocation information, a certification environment was built for the scenarios. The environment consisted of a root certificate authority (CA), which was used to sign an intermediate CA certificate. The intermediate CA signed certificates for queue managers, client applications, and WebSphere MQ Advanced Message Security (AMS) signing and encryption tasks.

In addition to the certificate authorities, an Online Certificate Status Protocol (OCSP) responder was built to respond to OCSP requests that relate to the intermediate CA.

A directory server was built and used to host Certificate Revocation Lists (CRLs) for the root and intermediate CAs.

Finally, a web server was built so that CRLs could be delivered independently of the server and file system where they were created. This server was used to script the download of CRLs from the intermediate CA so that they could be uploaded to the Lightweight Directory Access Protocol (LDAP) server.

8.2 Servers and network topology

Figure 8-1 on page 85 shows the topology of the servers that were used in the scenarios.

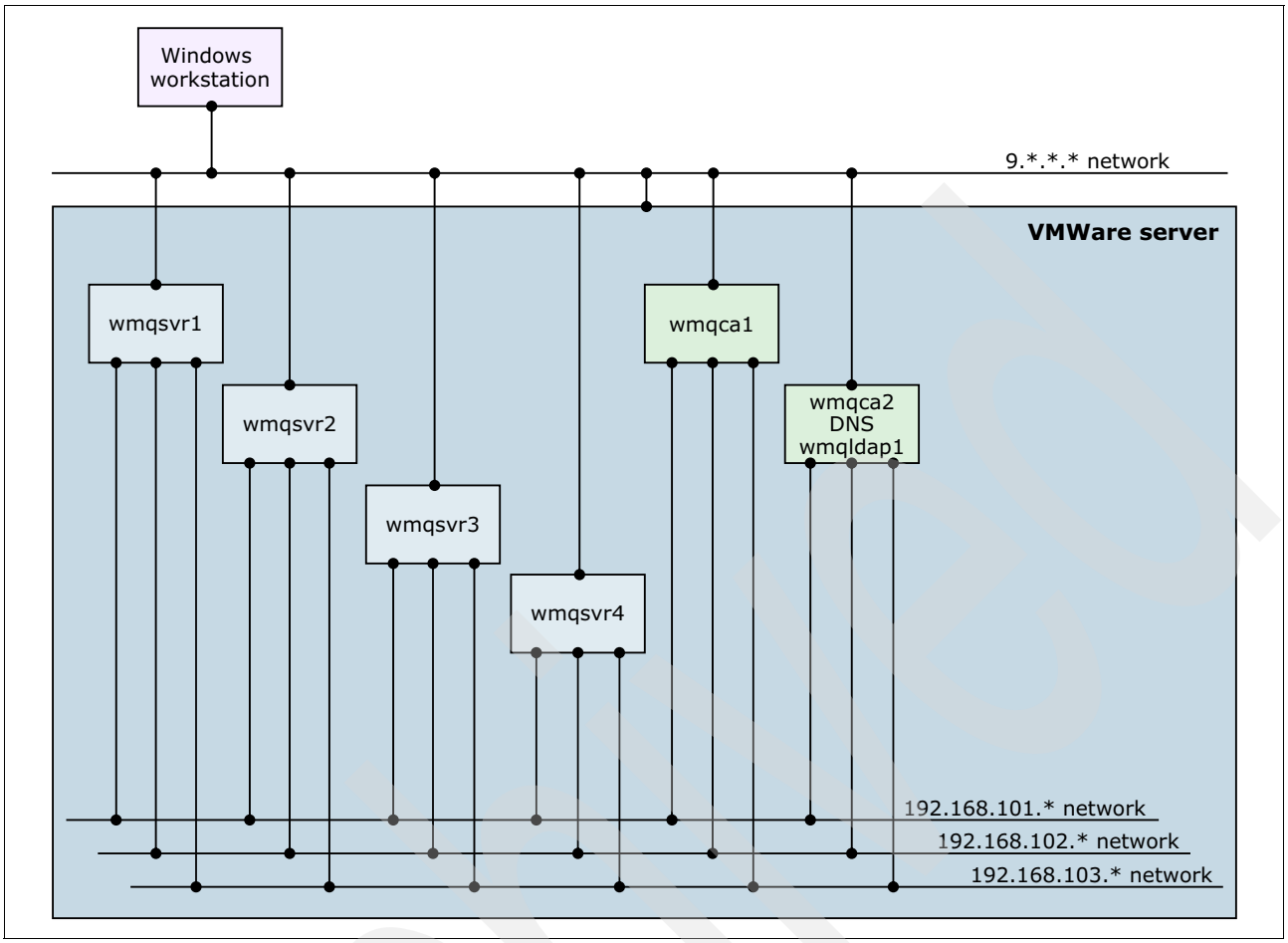


Figure 8-1 Topology of scenario servers and network

8.3 Operating systems and infrastructure software

All of the scenarios use WebSphere MQ queue managers running on Linux. This platform was chosen for ease of implementation in a virtual environment and because the members of the team working to create the book were familiar with it. The specific version of Linux is Red Hat Enterprise Linux (RHEL) version 6.3.

Two Windows Server 2008 R2 servers were used to host the certificate management infrastructure. These servers consisted of one server hosting Windows Active Directory Certificate Services (ADCS) running as a stand-alone root CA. A second server also ran ADCS, but it was an intermediate CA and used a certificate that was signed by the root CA.

The intermediate CA server was also configured with additional software to provide access to certificate revocation information:

- The OCSP Responder component of ADCS provided revocation via the Online Certificate Status Protocol (OCSP).
- An instance of Microsoft Internet Information Server (IIS) served the CRL files from a directory via HTTP.
- Microsoft Domain Name Service (DNS) was used to simplify host name management.
- An instance of IBM Tivoli Directory Server hosted CRLs so that WebSphere MQ could access them via Lightweight Directory Access Protocol (LDAP).

8.3.1 Virtualization

All of the servers that are referred to in the scenarios are virtual machines that are hosted on a single Intel server running VMWare ESXi version 5. Three virtual networks were created to simulate management, private and public data flows, and routing.

8.3.2 UNIX servers

All software products and versions that were used in the scenarios on UNIX servers are listed in Table 8-1.

Table 8-1 UNIX software

Software product	Installed version	Notes
Red Hat Enterprise Linux (RHEL)	6.3	64 bit.
IBM WebSphere MQ Server	7.5.0.0	For Linux 64 bit. All components installed.
IBM WebSphere MQ Internet pass-thru (MQIPT)	2.0.0.3	IBM SupportPac MS81.
IBM Java SE Runtime Environment	1.6.0	Part of the WebSphere MQ package.
Wireshark	1.2.15	Part of RHEL 6.3.
gcc	4.4.6	Part of RHEL 6.3.

Other than WebSphere MQ and MQIPT, the specifics of the software used are not critical. Equivalent scenarios could have been built using other operating systems or other versions of the associated software.

8.3.3 Windows servers

All software products and versions that are installed on Windows servers are listed in Table 8-2.

Table 8-2 Windows software

Software product	Installed version	Notes
Microsoft Windows Server	Windows Server 2008 R2 Enterprise	ADCS, DNS, and Web Server Roles are used.
IBM Tivoli Directory Server	6.2	
IBM WebSphere MQ Explorer	7.5.0.0	Free IBM SupportPac MS0T.
cURL	7.23.1	Downloaded from http://curl.askapache.com/download/curl-7.23.1-win64-ssl-sspi.zip.

There is nothing unique about the software packages that were used to implement the certificate authorities and associated infrastructure. Alternative software from other vendors or from Open Source Initiatives could have been used to implement equivalent facilities. Due to time constraints, it was necessary to choose one option for each of the services we needed, and implement with that option, even though it would have been interesting to demonstrate multiple options for this infrastructure software.

8.4 Operating system configuration

This section describes the common configuration tasks for the operating systems in all scenarios. This configuration includes the provisioning of resources, such as network, CPU, disk space, kernel parameters, user IDs, and groups.

8.4.1 Virtualization

All operating system images are virtual guests within a VMware ESXi v5 server. Each guest was provisioned with four network interfaces, as listed in Table 8-3.

Table 8-3 Network interfaces

Name	IP addresses	Purpose
eth0	192.168.101.*	Management interface.
eth1	192.168.102.*	Private data interface.
eth2	192.168.103.*	Public data interface.
eth3	9.*.*.*	Internal network interface. Allocated via Dynamic Host Configuration Protocol (DHCP).

For the purpose of illustrating all of the scenarios except business-to-business, the 192.168 interfaces are used. In the Business-to-Business (B2B) scenario, the connection from MQIPT to the external business partner is done by using the IBM network addresses for those sockets.

8.5 WebSphere MQ installation and configuration

This section describes the common configuration tasks for WebSphere MQ queue managers and resources that were performed for the scenarios in this book:

- WebSphere MQ v7.5.0.0 was installed with all supplied packages, except non-English message catalogs, on each of the UNIX systems.
- MQIPT was installed on the wmqiptsvr1 Linux server, which acted as the proxy server for our B2B scenario.

8.6 Other software installation and configuration

This section describes the common installation and configuration tasks for other common software that is used by the scenarios:

- Wireshark was installed on all UNIX systems to aid in problem determination. It allowed us to identify the underlying cause of connection failures when the messages reported by WebSphere MQ or MQIPT were ambiguous as to possible causes. Wireshark was installed using normal RHEL facilities (yum).
- cURL was installed on the CA server to facilitate the download of the CRL files from the web server.

8.7 Naming standards and conventions

The scenarios use generic names for all entities to avoid association with any real companies or configurations. They still employ the common practices and preferred methods of naming and identifying components of the middleware infrastructure.

8.7.1 Host names

Our hosts were configured in the private domain saw216.itso.ibm.com.

All names were prefixed with `wmq` and then named for the major function that they illustrated within the scenarios:

- wmqsvr1 through wmqsvr5: WebSphere MQ hosts
- wmqiptsvr1: MQIPT host
- wmqca1 through wmqca2: CA hosts (including OCSP and web server)
- wmqldap1: LDAP server for providing CRLs (alias of wmqca2)

The plain host name resolved to the private network address:

- Host names suffixed ‘-m’ resolved to the management address for that host.
- Host names suffixed ‘-e’ resolved to the public (external) interface address.

8.7.2 User ID and group names

The following conventions were used for user ID and group names:

- WebSphere MQ processes run with the mqm user ID and mqm group, which is the normal practice when using this product.
- Application account names were prefixed with "`mq`" and a value indicating usage, for example, `mqcust1` or `mqapp1`.
- Message Channel Agent users (MCAUSERs) were prefixed with "`mca`" and then a value indicating usage.
- MCAUSERs representing other queue managers were given the queue manager name in lowercase.

8.7.3 Queue manager names

Queue manager names were made as simple as possible, for example, `QM1` and `QM2` for application queue managers. `QMGW1` is the name for the gateway queue manager in the B2B scenario.

An important factor in any queue manager name is to keep it short (no more than eight characters) and to avoid characters that are going to cause name misspelling or channel naming ambiguity (such as '.'). Numbers and uppercase letters are preferred.

8.7.4 Channel names

Conventional channels are named *FromQM.ToQM*. In a more complex environment, channels may need an identifier to allow multiple channels between a pair of queue managers.

Cluster channels are named *ClusterName.ToQM*. Cluster channels are only used for a single cluster, even if the queue manager is a member of multiple clusters. This convention helps to simplify authentication and authorization settings for channels, and provides improved troubleshooting information in messages.

Server connection channels are named *Purpose.QM*.

8.7.5 WebSphere MQ object names

Objects were created with uppercase names. Since WebSphere MQ automatically shifts names to uppercase in the runmqsc environment unless they are quoted, using uppercase names helps to simplify administration and to avoid confusion between characters, such as lowercase l (L) with uppercase I (i).

8.8 Certificate authorities

Microsoft Windows Server 2008 R2 was used with the ADCS role to provide CA capability. Two servers were installed in a stand-alone configuration (no windows domain).

A root CA was configured on server wmqca1. The distinguished name (DN) of the root CA certificate was:

```
CN=WMQCA1 MQ Root CA,OU=SA-W216 Residency,O=IBM ITSO,C=US
```

The certificate was created using the sha256 signature algorithm.

The following primary reference was used for installing and configuring the root CA:

http://technet.microsoft.com/en-us/library/cc731183.aspx

An intermediate CA was built on server wmqca2. The intermediate CA certificate was signed by the root CA and used the sha256 signature algorithm.

The distinguished name (DN) of the root CA certificate was:

CN=WMQCA2 MQ Intermediate CA,OU=SA-W216 Residency,O=IBM ITSO,C=US

This primary reference was used for installing and configuring the intermediate CA. (The reference calls this CA a Subordinate CA.)

http://technet.microsoft.com/en-us/library/cc772192

The web server role, which installs Internet Information Server (IIS), was installed on wmqca2. This web server was used by the CA to allow for web-based certificate signing requests and also to serve the CRL files that are produced by the CAs via HTTP. The installation of IIS was guided by these instructions:

http://technet.microsoft.com/en-us/library/cc771209.aspx

8.9 OCSP responder

On wmqca2, the additional role services for OCSP responder (for use with stand-alone CAs as opposed to enterprise CAs) were installed. It was configured by using advice from the official Microsoft enterprise support blog:

http://blogs.technet.com/b/askds/archive/2009/06/24/implementing-an-ocsp-responder-part-i-introducing-ocsp.aspx

Authority Information Access (AIA) information was added to the certificates that are signed by the intermediate CA to reference this OCSP responder at URL http://wmqca2/ocsp.

8.10 LDAP server to host CRLs

On the wmqca2 server (alias wmqldap1), the IBM Tivoli Directory Server product was installed so that CRLs could be served via LDAP. The server was configured with a user ID that was enabled to update elements in the directory and a bind user ID to allow WebSphere MQ processes to download the CRLs.

A base DN was configured to host the CRL relative DNs (RDNs):

C=US,O=IBM ITSO

An organizational unit was created under the base DN:

C=US,O=IBM ITSO,OU=SA-W216 Residency

Bind users were created to allow WebSphere MQ to bind to the directory to retrieve CRL information and to allow the CA administrator to upload CRLs into the directory.

The CRL information was then added with the RDNs `CN=WMQCA1 MQ Root CA` and `CN=WMQCA2 MQ Intermediate CA` under the OU so that the DNs formed matched the DNs of the CA certificates.

For the download of CRLs to work with the WebSphere MQ AUTHINFO mechanism, the DN within the LDAP server at which the CRLs are located must exactly match the DN of the CA certificate.

In order for them to be useful, CRLs from CAs that sign user certificates need to expire quickly and be replaced with updated CRLs. They are commonly replaced at 12-hour intervals. The generation of these CRLs and their upload to the web server, LDAP server, and OCSP responder need to be automated in a real-world environment.

During the residency, the Windows batch file script that is shown in Example 8-1 was used together with the LDAP Data Interchange Format (LDIF) file that is shown in Example 8-2 to periodically issue a new CRL, download it via the web server, and upload it to the LDAP server.

Example 8-1 refreshldap.bat. script to download a CRL and update the LDAP copy

```
echo off
rem Download CRLs and certificate from CA.
rem upload to LDAP server.

c:
cd \wmqca2

del temp1.crl
del temp1delta.crl
del temp1.der

certutil -crl

c:\Software\curl -o temp1.crl
http://wmqca2/crl/WMQCA2%%20MQ%%20Intermediate%%20CA.crl
c:\Software\curl -o temp1delta.crl
http://wmqca2/crl/WMQCA2%%20MQ%%20Intermediate%%20CA+.crl
c:\Software\curl -o temp1.der http://wmqca2/crl/wmqca2.der
copy "c:\wmqca1\WMQCA1 MQ Root CA.crl" c:\wmqca2\temp0.crl
copy temp0.crl "CertEnroll\WMQCA1 MQ Root CA.crl"

"c:\Program Files\IBM\LDAP\V6.2\bin\ldapadd.cmd" -r -g -v -D CN=root -w
passsword123 -i tempcrl-ref.ldif
```

Example 8-2 tempcrl-ref.ldif LDIF file to load the updated CRL into an LDAP server

```
version: 1

dn: cn=WMQCA2 MQ Intermediate CA,ou=SA-W216 Residency,O=IBM ITSO,C=US
cn: WMQCA2 MQ Intermediate CA
#
objectclass: cRLDistributionPoint
objectclass: certificationAuthority
objectclass: pkiCA
certificateRevocationList;binary:< file:///c:\wmqca2\temp1.crl
deltaRevocationList;binary:< file:///c:\wmqca2\temp1delta.crl
authorityRevocationList;binary:< file:///c:\wmqca2\temp1.crl
```

```
caCertificate;binary:< file:///c:\wmqca2\temp1.der

dn: cn=WMQCA1 MQ Root CA,ou=SA-W216 Residency,O=IBM ITSO,C=US
cn: WMQCA1 MQ Root CA
#
objectclass: cRLDistributionPoint
objectclass: certificationAuthority
objectclass: pkiCA
certificateRevocationList;binary:< file:///c:\wmqca2\temp0.crl
deltaRevocationList;binary::
authorityRevocationList;binary:< file:///c:\wmqca2\temp0.crl
caCertificate;binary:< file:///c:\wmqca1\wmqca1.der
```

Although the exact details of server locations, files names, bind DNs, and passwords will be different for another implementation, the basics of periodically downloading a CRL from the certificate authority and loading it into the LDAP server and OCSP responder would be needed for any implementation of a CRL infrastructure for WebSphere MQ.

8.11 WebSphere MQ (CMS) keystores

WebSphere MQ and WebSphere MQ client applications use X.509 certificates for authentication via SSL/TLS. The private keys, personal certificates, and trusted certificates that WebSphere MQ uses for identity and validation are all stored in Certificate Management Services (CMS) keystores. CMS keystores are managed using IBM Global Secure ToolKit (GSKit). Where possible, the keys should be managed using Federal Information Processing Standard (FIPS) compliant tools.

This section provides sample command lines that can be used to perform useful functions that relate to WebSphere MQ keystores.

Creating a pseudo random passphrase

The following command will create a random string of characters and then remove the characters with which a UNIX shell might have problems. The command saves 64 characters in a file for use as the passphrase for a keystore:

```
runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65 > key.passwd
```

Creating a keystore

The following command will create a CMS keystore using the random passphrase stored in the `key.passwd` file. It will stash the passphrase so that other **runmqakm** commands and WebSphere MQ can access the keystore.

```
runmqakm -fips -keydb -create -db key.kdb -pw "`cat key.passwd`" -type cms -stash
-empty
```

Adding trusted certificates

The following command uses the **-add** option to add CA certificates and self-signed certificates that WebSphere MQ should trust to the keystore:

- `runmqakm -fips -cert -add -db key.kdb -stashed -file wmqca1.cer -label "WMQCA1 MQ Root CA" -trust enable`
- `runmqakm -fips -cert -add -db key.kdb -stashed -file wmqca2.cer -label "WMQCA2 MQ Intermediate CA" -trust enable`

Creating a certificate request

The following command will create a private key and a certificate signing request (CSR) file. The CSR is sent to a CA, which signs it and returns a certificate.

```
runmqakm -fips -certreq -create -db key.kdb -stashed -label ibmwebspheremqqm1 -dn
"CN=QM1,OU=SA-W216 Residency,O=IBM ITSO,C=US" -target qm1.req -sigalg sha256 -size
2048
```

Re-creating a certificate request to support certificate renewal

The following command will use the existing private key/public key pair in the CMS keystore to generate a new CSR. The new CSR can be signed, and the certificate imported, replacing the existing certificate. This process allows a certificate to be renewed before it expires with little disruption.

```
runmqakm -certreq -recreate -db key.kdb -stashed -label ibmwebspheremqqm1 -target
qm1.recreate.req -sig_alg sha256
```

Receiving signed certificates (original or renewed)

The following command will receive a certificate signed by a CA into the keystore and match it with the private key that is already in place. The private key is created when the CSR is originally created.

```
runmqakm -fips -cert -receive -db key.kdb -stashed -file qm1.cer
```

Creating a self-signed certificate

In some cases, self-signed certificates can be used instead of getting a certificate signing request signed by a certificate authority. A self-signed certificate is a certificate, which is signed by using its own private key. The following command will create a self-signed certificate:

```
runmqakm -fips -cert -create -db key.kdb -stashed -label ibmwebspheremqqm1 -dn
"CN=QM1,OU=SA-W216 Residency,O=IBM ITSO,C=US" -sigalg sha256 -size 2048
```

Extracting certificates

A certificate that is held in a CMS keystore could be needed for import into another keystore where it will provide trust. This certificate could be a self-signed certificate used with MQIPT or a certificate for AMS. In most cases, only self-signed certificates need to be exported because the signed certificate file from a CA can be kept in the keystore directory. The following command will extract a certificate from a keystore:

```
runmqakm -cert -extract -db key.kdb -stashed  -target qm1.cer -format ascii -label
ibmwebspheremqqm1
```

Exporting a private key and certificate to a keystore

When the same private key and certificate are needed on multiple servers, the keys can be exported from a CMS keystore. This export might also be required in an environment where private key material has to be stored in escrow or for backup purposes. The following command will export a private key and certificate from a CMS keystore to a PKCS#12 keystore. Although the same passphrase is used for both keystores, using the same passphrase is not necessary. A different passphrase could have been used for the two stores.

```
runmqakm -cert -export -db key.kdb -pw "`cat key.passwd`" -label ibmwebspheremqqm1
-target qm1.p12 -target_type pkcs12 -target_pw "`cat key.passwd`"
```

Importing a private key and certificate from a keystore to CMS

The following command will import a private key and certificate from a PKCS#12 keystore to CMS. The source of the certificate and key is the **-db** value. The source will be copied into the **-target** keystore. The object named in **-db** is not changed. If the label needs to be changed, add the **-new_label** option.

```
runmqakm -cert -import -db qm1.p12 -pw "`cat key.passwd`" -type pkcs12 -label
ibmwebspheremqqm1 -target key.kdb -target_type cms -target_pw "`cat key.passwd`"
```

Listing the certificates in a keystore

The following command will list the certificates in a keystore:

```
runmqakm -cert -list -db key.kdb -stashed
```

Showing the details of a certificate in a keystore

The following command will show the details of a certificate in a keystore:

```
runmqakm -cert -details -db key.kdb -stashed -label ibmwebspheremqqm1
```

Deleting a certificate from a keystore

This command can delete either a trusted or personal certificate. It will delete the private key if a private key is associated with the certificate. It will not prompt for confirmation so use this command carefully:

```
runmqakm -cert -delete -db key.kdb -stashed -label ibmwebspheremqqm1
```

Securing the SSL directory and keystore files

The following commands are used to secure the SSL directory and keystore files:

- `chmod 700 /var/mqm/qmgrs/QM1/ssl`
- `chmod 400 /var/mqm/qmgrs/QM1/ssl/*`

8.12 Other certificate tools

During this project, a number of tools were used to work with certificates, certificate stores, and other cryptographic entities. While the main tool was the **runmqakm** tool running in FIPS mode, as illustrated in 8.11, “WebSphere MQ (CMS) keystores” on page 92, we also used these tools:

- **runmqckm**, when dealing with Java Keystore (JKS) files
- **openssl**, for viewing certificate status returned via OCSP

runmqckm for Java Keystore manipulation

The **runmqckm** tool is similar to **runmqakm**, but it does not support the **-fips** or **-stashed** options. Importantly, it does support the additional keystore type, **jks**, for creating and working with JKSs.

Because the **-stashed** option is not available, the keystore passphrase is needed on the command line or **runmqckm** will prompt for it. On the command line, the option **-pw `cat key.passwd`** can be used to use the passphrase stored in a plaintext password file, `key.passwd`.

```
runmqckm -cert -details -db key.kdb -pw ‘cat key.passwd‘ -label ibmwebspheremqqm1
```

openssl OCSP function

The **openssl ocsp** command can be used to either validate that an OCSP responder is working, or that the specific certificate is still valid. The OCSP responder is named in the **-url** argument. The certificate that is queried and the signer certificate are provided in the **-cert** and **-issuer** arguments respectively.

For more information about this command, see this website:

http://www.openssl.org/docs/apps/ocsp.html\

9

Scenario: WebSphere MQ administration

This scenario demonstrates the configuration of a queue manager for secure remote administrative access using Transport Layer Security (TLS)/Secure Sockets Layer (SSL) certificate authentication (CA) and encryption over client channels. Any queue manager that is administered by using client-based tools is a candidate for this scenario.

This chapter contains the following topics:

- Scenario overview
- Implementing the scenario
- Configuring WebSphere MQ Explorer for the anonymous administration role
- Configuring WebSphere MQ Explorer for a limited administration role
- Configuring WebSphere MQ Explorer for a full administration role
- Summary

9.1 Scenario overview

As of IBM WebSphere MQ version 7.1, the default settings of channel authorities on a new queue manager do not allow anonymous remote client connections. The base security policy is *deny-all* with explicitly enumerated exceptions.

There are two cases in which this scenario might not apply:

- The queue manager is administered through an adjacent queue manager. As a rule, it is not a good practice to allow administration from adjacent queue managers so this case is quite rare.
- All WebSphere MQ administration is performed locally through the command line. This case is not typical but is sometimes employed where security requirements or policies prevent remote connections.

This scenario has the following objectives:

- Provision strongly authenticated administrative access between the queue manager and a remote IBM WebSphere MQ Explorer client.
- Provision non-administrative anonymous access for interactive users.
- Enforce that non-administrators cannot obtain administrative rights.
- Illustrate the precedence of channel authentication (CHLAUTH) rules.

To achieve these objectives, the scenario will use the following security controls:

- WebSphere MQ:
  - Strong authentication of administrators using X.509 certificates on TLS/SSL enabled channels.
  - CHLAUTH rules to map distinguished name (DN) to MCAUSER.
  - CHLAUTH rules to restrict administrative access to specific channels.
  - **SET AUTHREC** commands to restrict access of unauthenticated users.
- Policy:
  - Strongly authenticate all administrative access.
  - Unauthenticated users can display object names and their attributes.
  - No administration from adjacent queue managers.
  - CHLAUTH rules enabled.
  - Full WebSphere MQ administrators are in the mqm (or platform equivalent) group.

Chapter 6, “WebSphere MQ security controls” on page 41 describes controls that can be used for WebSphere MQ security and includes some techniques for implementing administration controls. This chapter puts administration controls into action and provides in-depth information and discussion. It presents preferred methods and options with considerations for risk, cost, practicality, and business requirements.

The default installation of WebSphere MQ version 7.0.1 and older versions contained wide open access to all administration functions from anonymous remote locations. Simple steps could be taken to provided limited controls or prevent this access but these steps were often not given proper consideration, with potentially disastrous results. If you are migrating to version 7.1 or later with existing queue managers, CHLAUTH profiles will need to be defined with policies to provide base controls on administrative access. Strong TLS/SSL two-way authentication and encryption should also be implemented on client channels that are used for remote administration.

The ultimate goal is to completely restrict all administrative functions in WebSphere MQ to authenticated individuals and applications, limit them to role-based authorized functions, and provide accountability for their actions.

9.1.1 Scenario design

This scenario is designed to demonstrate the simplest implementation of role-based remote administration of a WebSphere MQ queue manager using secure client connections. A single queue manager runs on one server, WebSphere MQ Explorer runs on a Microsoft Windows workstation, and they are linked via a TCP/IP network.

There are other possible administration topologies and advanced security techniques that could be used, such as firewalls, administration networks, dedicated port listeners, and multiple queue managers, but they are beyond the objectives of this scenario. The principles in this scenario can be applied to securing other remote WebSphere MQ administration tools that use client channels.

The diagram in Figure 9-1 shows WebSphere MQ Explorer connected to the queue manager via a client channel. Both systems have TLS/SSL certificate keystores. The certificates have been signed by a certificate authority (CA).

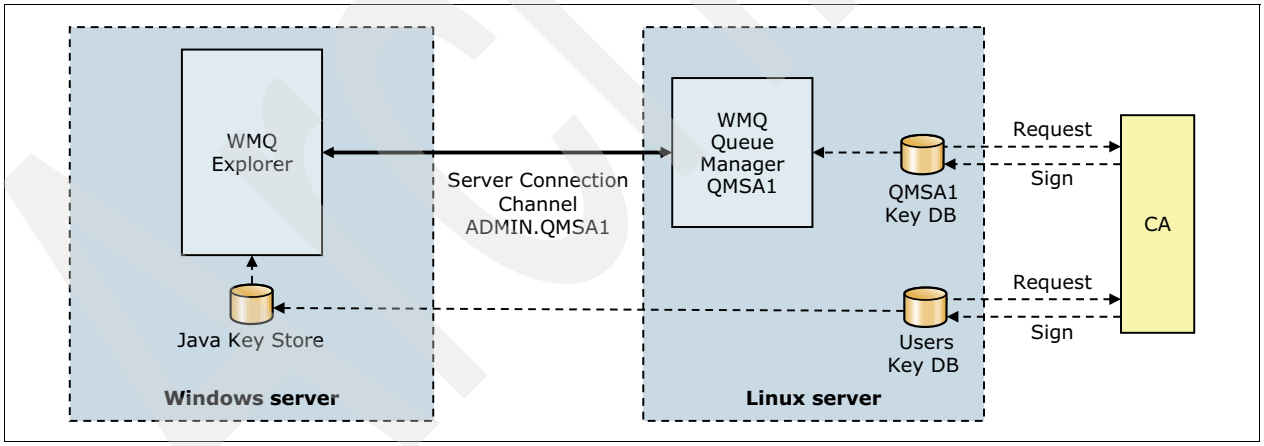


Figure 9-1 Explorer and queue manager topology in secure WebSphere MQ administration scenario

The queue manager is named `QMSA1` on Linux server `wmqsvr1`.

Certificate revocation checking can be implemented on all certificates in this scenario but it has been omitted for clarity and simplicity. In practice, it is not usually necessary or appropriate for the client to check revocation of the queue manager's certificate. Revocation is covered in detail in Chapter 12, "Scenario: CRL/OCSP certificate revocation" on page 219.

There are various combinations of TLS/SSL certificate implementation for securing WebSphere MQ Explorer access to a queue manager. See a post on the Hursley blog for WebSphere MQ:

http://hursleyonwmq.wordpress.com/2007/07/30/using-websphere-mq-explorer-with-ssl/

Another post describes using WebSphere MQ Explorer as a read-only viewer:

http://hursleyonwmq.wordpress.com/2007/02/08/using-websphere-mq-explorer-as-a-read-only-viewer/

These posts were written for WebSphere MQ version 6.0 so they do not mention the channel authentication (CHLAUTH) rules that became available in version 7.1.

9.1.2 Prerequisites

The scenario has the following prerequisites:

- WebSphere MQ Server

  A licensed and supported version of IBM WebSphere MQ Server version 7.1.0.0 or later on a supported server platform. This scenario uses version 7.5.0.0 for RHEL Linux 6.2 operating system on X86-64 compatible hardware.

- WebSphere MQ Explorer

  A licensed and supported version of IBM WebSphere MQ Explorer version 7.1.0.0 or later on a supported workstation platform. This scenario uses version 7.5.0.0 of IBM SupportPac MS0T for the Windows XP Professional SP3 operating system on X86-32 compatible hardware.

  IBM SupportPac MS0T is a stand-alone packaging of IBM WebSphere MQ Explorer that is freely available for download and installation from the IBM Internet website:

  http://www-01.ibm.com/support/docview.wss?uid=swg24021041

  Downloads are available for the currently supported WebSphere MQ version 7.0.1.8, where Eclipse version 3.3.1 is a prerequisite, and version 7.1.0.1, where Eclipse is included. It can run on supported Windows 32-bit and 64-bit operating systems and supported Linux 32-bit and 64-bit operating systems.

  The SupportPac does not require WebSphere MQ Server or Client to be separately installed on the WebSphere MQ Explorer workstation.

- Certificate authority

  Certificate authorities (CA) are available that can be used to create CA-signed private certificates. The setup of the CA for scenarios in this book is described in 8.8, “Certificate authorities” on page 89.

The following assumptions are made about the environment and the user:

- The server and workstation have correctly licensed, installed, and configured operating systems and prerequisite software.
- The server and workstation are on a TCP/IP network and they can communicate with each other via unique IP addresses.
- The user of this scenario is familiar with UNIX and Linux shell commands and WebSphere MQ administration on UNIX and Linux, and the user is also familiar with Windows and using WebSphere MQ Explorer.

- The user of this scenario has WebSphere MQ administrator access to the queue manager Linux server to create and configure a dedicated queue manager. The user can request the creation of user IDs, groups, and group memberships on the server.
- The user of this scenario has a user ID on the WebSphere MQ Explorer Windows workstation that is not the same as any user ID on the queue manager server. The user ID should not be a Windows Administrator or have any other special rights or group memberships. This assumption is not a requirement for the scenario but ensures that elevated authorities do not influence its operation.

9.1.3 Using the additional material for scripts and common variables

Most of the Linux and WebSphere MQ administration commands in this scenario are provided in shell scripts to save a lot of typing, copying, and pasting from this book. The scripts are all available in the additional materials for this book. For information about getting these materials, see Appendix D, “Additional material” on page 341.

Extracting the scripts

Most of the scripts can be run by any WebSphere MQ administrator user, but some scripts for certificate manipulation need to be run by the `mqm` user ID. For this scenario, it is easier to extract all the scripts in the `mqm` home directory and run them from the `mqm` user ID. Follow these steps:

1. Log on or **`su`** to the `mqm` user ID on the `wmqsvr1` server.
2. Make a new sub-directory in the `mqm` home directory, for example `ch9admin`.
3. Change to the sub-directory and transfer in the file `ch9admin.tar`.
4. Extract all the files from the tar file by using the shell command **`tar -xvf ch9admin.tar`**, which also creates the `certs` sub-directory that contains files.

To run the scripts, your current directory needs to be set to the directory containing the scripts. The preferred method of running the scripts is to enter `./` followed by the file name, for example, **`./createqmgr.ksh`**.

Editing the common variables

Edit the **`vars.ksh`** script file by using **`vi`** or your favorite text editor to view the values of common variables that are sourced by other scripts from this file, as shown in Example 9-1. You can change these values if they do not suit the environment in which you are implementing the scenario.

Example 9-1 Common variables in vars.ksh script

```
QMGR="QMSA1"          # Local queue manager name
HOST="9.42.170.149"   # Host name of queue manager
PORT="14151"          # TCP listener port number of queue manager
```

9.2 Implementing the scenario

This section discusses the following steps to implement the scenario:

- Preparing the operating system user IDs and groups
- Creating the queue manager and listener
- Authorizing queue manager and system objects to enable remote WebSphere MQ Explorer
- Defining application objects and limited administration authority
- Providing authority to display all objects
- Defining a channel for anonymous remote WebSphere MQ Explorer
- Defining a secure channel for remote administration roles
- Creating a key repository for queue manager
- Generating the queue manager certificate and adding it to the key repository
- Creating a key repository for users
- Generating the user certificates and adding them to the key repository
- Building the Java keystore files for users of WebSphere MQ Explorer
- Setting up the WebSphere MQ Explorer workstation

Executing scripts: All the scripts and actions taken in this chapter are done on the queue manager server unless otherwise noted.

9.2.1 Preparing the operating system user IDs and groups

Specific user IDs and groups are used to execute the scenario. This section describes how they were created.

Queue manager server

Dedicated user IDs and groups are created on the queue manager server (`wmqsvr1`) to identify and control administrative roles. The use of the `mqm` user ID is avoided when considering the security of remote administrative access. The `mqm` user ID should only be asserted by processes that run on the queue manager's server. Example 9-2 contains the commands that are run by the `root` user to create user IDs and groups for the three types of administrative roles that are demonstrated in this scenario. You can request the system administrator to run the script, or if you know the root password, a shell command, such as **`su - -c "/home/mqm/ch9admin/usersgroups.ksh"`**, can be run.

Example 9-2 Create user IDs and groups in usersgroups.ksh

```
#!/usr/bin/ksh
# Run this script from root
useradd -c "MQ full admin user" -g mqm -G mqm mqadm1

groupadd mqappg1
useradd -c "MQ app1 admin user" -G mqappg1 mqapp1

groupadd mqanong
useradd -c "MQ anon user" -G mqanong mqanon
```

These commands use the system defaults for user ID account expiry, inactive disablement, primary group, and logon shell. In an actual situation, you might consider setting these attributes so that the user IDs do not expire or become inactive, do not have a home directory, cannot log on to a shell, and do not have a usable password. For example, on RHEL Linux, use **-e 2099-12-31 f -1 -M -s /bin/nobody**. WebSphere MQ only uses these user IDs as authorization entities, it does not use their password, home directory, or logon capabilities.

WebSphere MQ Explorer workstation

A non-administrative user ID was created to run WebSphere MQ Explorer on the Windows workstation. The user ID did not also exist on the `wmqsvr1` queue manager server.

9.2.2 Creating the queue manager and listener

Create a new queue manager and listener by running the script that is shown in Example 9-3. With this script, small circular recovery logs are created to conserve disk space and the following basic security hardening steps are taken:

- A dead-letter queue (DLQ) is assigned to the queue manager, using the default `SYSTEM.DEAD.LETTER.QUEUE` local queue that is defined when the queue manager is created.
- All queue manager events that relate to security are enabled.
- All certificates must be Federal Information Processing Standard (FIPS)-compliant.

Example 9-3 Create queue manager using createqmgr.ksh script

```
#!/usr/bin/ksh
. ./vars.ksh

echo "*** Create queue manager $QMGR with DLQ and small circular recovery logs"
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE -lc -lp 2 -ls 2 -ls 100 $QMGR

echo "*** Start the queue manager"
strmqm $QMGR

echo "*** Enable all security related events
alter qmgr authorev(enabled) inhibtev(enabled) localev(enabled) +
 remoteev(enabled) strstpev(enabled) chlev(enabled) chadev(enabled) +
 sslev(enabled) perfmev(enabled) configev(enabled) cmdev(enabled)

*** Require certificates to be FIPS compliant
alter qmgr sslfips(yes)

*** Disable all incoming system channels
alter channel('SYSTEM.AUTO.RECEIVER') chltype(rcvr) +
  mcauser('*NOACCESS') maxmsgl(1)
alter channel('SYSTEM.AUTO.SVRCONN') chltype(svrconn) +
  mcauser('*NOACCESS') maxmsgl(1)
alter channel('SYSTEM.DEF.CLUSRCVR') chltype(clusrcvr) +
  mcauser('*NOACCESS') maxmsgl(1)
alter channel('SYSTEM.DEF.RECEIVER') chltype(rcvr) +
  mcauser('*NOACCESS') maxmsgl(1)
alter channel('SYSTEM.DEF.REQUESTER') chltype(rqstr) +
  mcauser('*NOACCESS') maxmsgl(1)
alter channel('SYSTEM.DEF.SERVER') chltype(svr) +
  mcauser('*NOACCESS') maxmsgl(1)
```

```
alter channel('SYSTEM.DEF.SVRCONN') chltype(svrconn) +
  mcauser('*NOACCESS') maxmsgl(1)

*** Define listener object for TCP port and start it
define listener('TCP.$PORT') trptype(tcp) replace +
 port($PORT) control(qmgr)
start listener('TCP.$PORT')
" | runmqsc $QMGR
```

This step assumes that some queue manager defaults are in place:

- Channel Authentication is enabled: `CHLAUTH(ENABLED)`.
- A default CHLAUTH rule blocks administrative access on all SVRCONN channels.
- Certificate keystores are located in the default directory and file name prefix:

  `SSLKEYR('/var/mqm/qmgrs/QMSA1/ssl/key')`

9.2.3 Authorizing queue manager and system objects to enable remote WebSphere MQ Explorer

Groups that are used for remote WebSphere MQ administration roles require the following authorities:

- Connect, inquire, and display authority to the queue manager object
- Put, inquire, and display authority to the system command queue

Groups that are used by WebSphere MQ Explorer users also require get, inquire, and display authority to the WebSphere MQ Explorer reply model queue. Run the script that is shown in Example 9-4 to provide the minimum authorities for the scenario. The script calls a function setauthrecadm to issue MQSC commands **SET AUTHREC**. It also contains an equivalent function setmqautadm that uses **setmqaut** program commands. The variables are set to the values that are shown in Example 9-1 on page 101.

Example 9-4 Authorizations enable remote WebSphere MQ Explorer operation using setadmauths.ksh

```
#!/usr/bin/ksh
. ./vars.ksh

# Function to provide group authority for admin and explorer
# using setmqaut MQ administration program
# Verifies by displaying authority for a userid in the group
setmqautadm() {
# Set authority to use queue manager
setmqaut -m $QMGR -t qmgr -g $1 +inq +connect +dsp
dspmqaut -m $QMGR -t qmgr -p $2
# Set authority to use command queue
setmqaut -m $QMGR -t q -n $CMDQ -g $1 +put +inq +dsp
dspmqaut -m $QMGR -t q -n $CMDQ -p $2
# Set authority to use explorer model queue
setmqaut -m $QMGR -t q -n $EXPM -g $1 +get +inq +dsp
dspmqaut -m $QMGR -t q -n $EXPM -p $2
}

# Function to provide group authority for admin and explorer
# using MQSC commands
# Verifies by displaying authority for a userid in the group
```

```
setauthrecadm() {
echo "
* Set authority to use queue manager
set authrec objtype(qmgr) group('$1') authadd(inq,connect,dsp)
* Set authority to use command queue
set authrec profile('$CMDQ') objtype(queue) group('$1') authadd(put,inq,dsp)
* Set authority to use explorer model queue
set authrec profile('$EXPM') objtype(queue) group('$1') authadd(get,inq,dsp)
" | runmqsc $QMGR
# Display effective authority for a userid in the group
dspmqaut -m $QMGR -t qmgr -p $2
dspmqaut -m $QMGR -t q -n $CMDQ -p $2
dspmqaut -m $QMGR -t q -n $EXPM -p $2
}

CMDQ="SYSTEM.ADMIN.COMMAND.QUEUE"
EXPM="SYSTEM.MQEXPLORER.REPLY.MODEL"

echo "*** Set authorities for application admin group mqappg1"
setauthrecadm mqappg1 mqapp1

echo "*** Set authorities for anonymous admin group mqanong"
setauthrecadm mqanong mqanon
```

The output of the script should display the authorities in Example 9-5.

Example 9-5 Output of setadmauths.ksh

```
Entity mqanon has the following authorizations for object QMSA1:
        inq
        connect
        dsp
Entity mqanon has the following authorizations for object
SYSTEM.ADMIN.COMMAND.QUEUE:
        put
        inq
        dsp
Entity mqanon has the following authorizations for object
SYSTEM.MQEXPLORER.REPLY.MODEL:
        get
        inq
        dsp
```

9.2.4 Defining application objects and limited administration authority

In this step, local queue, alias queue, and process objects that are used by two applications `APP1` and `APP2` are defined on the queue manager. Authority is set up to allow user IDs in the `mqappg1` group to perform administration operations on objects that are owned by the `APP1` application, but they do not have any administration access to objects that are owned by the `APP2` application.

The following administration operations are for queue objects:

- Get messages.
- Browse messages.

- Put messages.
- Inquire on some attributes.
- Set some attributes.
- Change all attributes.
- Display object name.
- Clear all messages from the queue.

The following administration operations are for process objects:

- Inquire on some attributes.
- Set some attributes.
- Change all attributes.
- Display object name.

Run the script in Example 9-6 to create the objects and define the limited authorities for the application group, `mqappg1`.

Example 9-6 Define application queues and authority using defappobjs.ksh

```
#!/usr/bin/ksh
. ./vars.ksh

echo "*** Define application objects"
echo "
* Application APP1
define qlocal('APP1.ABC.REQUEST') replace
define qlocal('APP1.ABC.REPLY') replace
define qlocal('APP1.EFG.REQUEST') replace
define qlocal('APP1.EFG.REPLY') replace
define qalias('APP1.HIJ.REQUEST') target('APP1.ABC.REQUEST') replace
define qalias('APP1.LMN.REQUEST') target('APP1.ABC.REQUEST') replace
define process('APP1.ABC.PROCESS') replace +
 appltype(unix) applicid('/usr/app1/abcprocess')
* Application APP2
define qlocal('APP2.XYZ.REQUEST') replace
define qlocal('APP2.XYZ.REPLY') replace
define qalias('APP2.UVW.REQUEST') target('APP2.XYZ.REQUEST') replace
define process('APP2.XYZ.PROCESS') replace +
 appltype(unix) applicid('/usr/app2/xyzprocess')
* Set up limited administration authority for an application group
set authrec profile('APP1.**') objtype(queue) group('mqappg1') +
  authadd(get,browse,put,inq,set,chg,dsp,clr)
set authrec profile('APP1.**') objtype(process) group('mqappg1') +
  authadd(inq,set,chg,dsp)
" | runmqsc $QMGR

echo "*** Display authority for associated userid"
dspmqaut -m $QMGR -t queue   -n APP1.ABC.REQUEST -p mqapp1
dspmqaut -m $QMGR -t process -n APP1.ABC.PROCESS -p mqapp1
```

The output of the script should display the authorities in Example 9-7.

Example 9-7 Output of defappobjs.ksh

```
Entity mqapp1 has the following authorizations for object APP1.ABC.REQUEST:
        get
        browse
        put
        inq
        set
        chg
        dsp
        clr
Entity mqapp1 has the following authorizations for object APP1.ABC.PROCESS:
        inq
        set
        chg
        dsp
```

9.2.5 Providing authority to display all objects

Remote administration tools usually enumerate and display the names of all objects of a particular type. This action requires authority to display all objects of that type, including the SYSTEM objects. The following script is based on the authorities that are provided by the Add Role-Based Authorities dialog in WebSphere MQ Explorer, which provides display access to the following objects:

- All queues
- Topics
- Channels
- Processes
- Name lists
- Authority information
- Client connection channels
- Listeners
- Services
- Communication information objects

Run the script in Example 9-8 to set up the authorizations for users of WebSphere MQ Explorer.

Example 9-8 Authorizations to display all objects using setdspauths.ksh

```
#!/usr/bin/ksh
. ./vars.ksh

# Function to provide display authority on all objects in all object types
# using setmqaut MQ administration program
setmqautdsp() {
setmqaut -m $QMGR -n "**" -t queue    -g $1 +dsp
setmqaut -m $QMGR -n "**" -t topic    -g $1 +dsp
setmqaut -m $QMGR -n "**" -t channel  -g $1 +dsp
setmqaut -m $QMGR -n "**" -t process  -g $1 +dsp
setmqaut -m $QMGR -n "**" -t namelist -g $1 +dsp
setmqaut -m $QMGR -n "**" -t authinfo -g $1 +dsp
setmqaut -m $QMGR -n "**" -t clntconn -g $1 +dsp
setmqaut -m $QMGR -n "**" -t listener -g $1 +dsp
setmqaut -m $QMGR -n "**" -t service  -g $1 +dsp
```

```
setmqaut -m $QMGR -n "**" -t comminfo -g $1 +dsp
}

# Function to provide display authority on all objects in all object types
# using MQSC commands
setauthrecdsp() {
echo "
set authrec profile('**') objtype(queue)    group('$1') authadd(dsp)
set authrec profile('**') objtype(topic)    group('$1') authadd(dsp)
set authrec profile('**') objtype(channel)  group('$1') authadd(dsp)
set authrec profile('**') objtype(process)  group('$1') authadd(dsp)
set authrec profile('**') objtype(namelist) group('$1') authadd(dsp)
set authrec profile('**') objtype(authinfo) group('$1') authadd(dsp)
set authrec profile('**') objtype(clntconn) group('$1') authadd(dsp)
set authrec profile('**') objtype(listener) group('$1') authadd(dsp)
set authrec profile('**') objtype(service)  group('$1') authadd(dsp)
set authrec profile('**') objtype(comminfo) group('$1') authadd(dsp)
" | runmqsc $QMGR
}

echo "*** Set authority for anonymous admin group to display all objects"
setauthrecdsp mqanong

echo "*** Set authority for application admin group to display all objects"
setauthrecdsp mqappg1
```

9.2.6 Defining a channel for anonymous remote WebSphere MQ Explorer

An objective of this scenario is to allow anonymous remote usage of WebSphere MQ Explorer to perform very restricted operations on the queue manager, such as displaying object names and their attributes. In an actual client environment, this usage might be further restricted or not provided, depending on your business requirements for administration and monitoring. WebSphere MQ Explorer connects to the queue manager via a client channel. The default channel name in the configuration dialog is `SYSTEM.ADMIN.SVRCONN`. This channel is defined on the queue manager with a fixed MCAUSER of `mqanon` that cannot be overridden by the client application and is not overridden by any Channel Authentication (CHLAUTH) rules. WebSphere MQ Explorer will use the authority of the `mqanong` group to perform Programmable Command Format (PCF) commands. Run the **`defanonsvrconn.ksh`** script in Example 9-9 to define the new client channel.

Example 9-9 Define channel for anonymous remote usage of WebSphere MQ Explorer

```
#!/usr/bin/ksh
. ./vars.ksh

echo "*** Define default channel for anonymous admin users"
echo "
define channel('SYSTEM.ADMIN.SVRCONN') chltype(svrconn) +
 mcauser('mqanon') +
 maxmsgl(4194304) +
 sslciph(' ') +
 sslpeer(' ') +
 descr('Anonymous MQ administrator')
" | runmqsc $QMGR
```

When a queue manager is created in WebSphere MQ version 7.1 and later, there are three CHLAUTH rules that are automatically defined. One of these rules relates to the SYSTEM.ADMIN.SVRCONN channel, as set using the MQSC command:

```
SET CHLAUTH('SYSTEM.ADMIN.SVRCONN') TYPE(ADDRESSMAP) +
DESCR('Default rule to allow MQ Explorer access') +
ADDRESS(*) USERSRC(CHANNEL) WARN(NO)
```

This command allows the channel to run from any IP address with any MCAUSER that is specified on the channel. This rule is required to override the following automatically defined rule that prevents any channel from running that has a name beginning with SYSTEM.:

```
SET CHLAUTH('SYSTEM.*') TYPE(ADDRESSMAP) +
DESCR('Default rule to disable all SYSTEM channels') +
ADDRESS('*') USERSRC(NOACCESS) WARN(NO)
```

9.2.7 Defining a secure channel for remote administration roles

A key objective of this scenario is to provide a secure channel for strongly authenticated remote clients to perform role-based administration of the queue manager. This function requires two-way TLS/SSL authentication so that the queue manager verifies the client's certificate. The queue manager can then perform SSL peer matching on the client's DN in their certificate and set the channel MCAUSER based on the Common Name (CN).

The scenario demonstrates two administration roles:

- Full administration rights, as provided by mqm group membership. The scenario uses the mqadm1 user ID for this role. The user ID should not be used for any other purposes on the system.
- Limited administration rights, as provided by membership of a group that has been granted particular administration authorities to particular objects. The scenario uses the mqapp1 user ID and mqappg1 group for this role. The user ID and group should not be used for any other purposes on the system.

The WebSphere MQ administrator connects via a dedicated client channel named ADMIN.QMSA1. This SVRCONN channel is defined on the queue manager with an MCAUSER of *NOACCESS. For each client connection to this channel, the actual MCAUSER is mapped using CHLAUTH rules.

This scenario uses cipher specification TLS_RSA_WITH_AES_128_CBC_SHA, which is FIPS-compliant. This specification is not the strongest specification available. You may want to use a stronger specification based on your business security requirements.

The SSLPEER attribute of the channel provides a DN that is matched against the DN of the client's certificate. This matching provides an additional verification to the DN checking that is done by CHLAUTH rules for SSLPEERMAP on the channel.

CHLAUTH rules are defined for the client administration channel to map the full DN to a specified user ID to be used as MCAUSER while the channel instance is running. In WebSphere MQ version 7.1 and version 7.5, a separate rule needs to be defined for each client's unique DN. Every personal user should have their own client certificate and unique DN.

The CHLAUTH rules also provide a check on the IP address range of the client.

Finally, a CHLAUTH rule needs to be defined for the channel to override the following rule that is automatically defined when a queue manager is created in WebSphere MQ version 7.1 and later. The default rule blocks all WebSphere MQ administrator user IDs on all client channels, as set using the MQSC command:

```
SET CHLAUTH('*') TYPE(BLOCKUSER) +
DESCR('Default rule to disallow privileged users') +
USERLIST(*MQADMIN) WARN(NO)
```

Run the **defsecsvrconn.ksh** script in Example 9-10 to define a new server connection channel and channel authentication rules.

Example 9-10 Define channel and authorization for secure remote administrator

```
#!/usr/bin/ksh
. ./vars.ksh

echo "
*** Secure MQ administration client channel with roles
*** MCAUSER is overridden by CHLAUTH rules
define channel('ADMIN.$QMGR') chltype(svrconn) trptype(tcp) replace +
 descr('Secure MQ administrator') +
 maxmsgl(4194304) +
 mcauser('*NOACCESS') +
 sslciph('TLS_RSA_WITH_AES_128_CBC_SHA') +
 sslpeer('OU=SA-W216 Residency,O=IBM ITSO,C=US') +
 sslcauth(required)

*** Define 2 new CHLAUTH rules for the secure MQ admin client
*** to map client DNs to MCAUSERs

set chlauth('ADMIN.$QMGR') type(sslpeermap) +
 action(replace) +
 descr('Full MQ admin user') +
 sslpeer('CN=mqadm1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
 address('9.42.*') +
 usersrc(map) mcauser('mqadm1')

set chlauth('ADMIN.$QMGR') type(sslpeermap) +
 action(replace) +
 descr('Application 1 MQ admin user') +
 sslpeer('CN=mqapp1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
 address('9.42.*') +
 usersrc(map) mcauser('mqapp1')

*** Define new channel authorization rule to override
*** default rule CHLAUTH(*) TYPE(BLOCKUSER) USERLIST(*MQADMIN)
*** The USERLIST can be any user that is not a valid user of the channel
set chlauth('ADMIN.$QMGR') type(blockuser) +
 action(replace) +
 descr('Rule to allow MQ admin userids on this channel') +
 userlist('nobody')
" | runmqsc $QMGR
```

The script in Example 9-11 can be run to display all the CHLAUTH records in the queue manager. An optional command-line argument allows the display to be limited to CHLAUTH record names beginning with a name.

Example 9-11 Display CHLAUTH records using dischlauth.ksh

```
#!/usr/bin/ksh
. ./vars.ksh
echo "dis chlauth(${1}*) all" | runmqsc $QMGR
```

The display should look like the output that is shown in Example 9-12.

Example 9-12 Output of dischlauth.ksh

```
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMSA1.
1 : dis chlauth(*) all
AMQ8878: Display channel authentication record details.
   CHLAUTH(ADMIN.QMSA1)                    TYPE(SSLPEERMAP)
   DESCR(Application 1 MQ admin user)      CUSTOM( )
   SSLPEER(CN=mqapp1,OU=SA-W216 Residency,O=IBM ITSO,C=US)
   ADDRESS(9.42.*)                         MCAUSER(mqapp1)
   USERSRC(MAP)                            WARN(NO)
   ALTDATE(2012-07-17)                     ALTTIME(11.47.23)
AMQ8878: Display channel authentication record details.
   CHLAUTH(ADMIN.QMSA1)                    TYPE(SSLPEERMAP)
   DESCR(Full MQ admin user)               CUSTOM( )
   SSLPEER(CN=mqadm1,OU=SA-W216 Residency,O=IBM ITSO,C=US)
   ADDRESS(9.42.*)                         MCAUSER(mqadm1)
   USERSRC(MAP)                            WARN(NO)
   ALTDATE(2012-07-17)                     ALTTIME(11.47.23)
AMQ8878: Display channel authentication record details.
   CHLAUTH(ADMIN.QMSA1)                    TYPE(BLOCKUSER)
   DESCR(Rule to allow MQ admin userids on this channel)
   CUSTOM( )                               USERLIST(nobody)
   WARN(NO)                                ALTDATE(2012-07-17)
   ALTTIME(11.47.23)
AMQ8878: Display channel authentication record details.
   CHLAUTH(SYSTEM.ADMIN.SVRCONN)           TYPE(ADDRESSMAP)
   DESCR(Default rule to allow MQ Explorer access)
   CUSTOM( )                               ADDRESS(*)
   MCAUSER( )                              USERSRC(CHANNEL)
   WARN(NO)                                ALTDATE(2012-07-17)
   ALTTIME(10.19.17)
AMQ8878: Display channel authentication record details.
   CHLAUTH(SYSTEM.*)                       TYPE(ADDRESSMAP)
   DESCR(Default rule to disable all SYSTEM channels)
   CUSTOM( )                               ADDRESS(*)
   MCAUSER( )                              USERSRC(NOACCESS)
   WARN(NO)                                ALTDATE(2012-07-17)
   ALTTIME(10.19.17)
AMQ8878: Display channel authentication record details.
   CHLAUTH(*)                              TYPE(BLOCKUSER)
   DESCR(Default rule to disallow privileged users)
   CUSTOM( )                               USERLIST(*MQADMIN)
   WARN(NO)                                ALTDATE(2012-07-17)
```

```
   ALTTIME(10.19.17)
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

9.2.8 Creating a key repository for queue manager

A WebSphere MQ queue manager uses a key database (KDB) in Certificate Management Services (CMS) format to store and manage certificates for TLS/SSL-enabled channels. The KDB files are also used to stash the KDB password in encrypted format, manage a Certificate Revocation List (CRL), and manage certificate requests. The location of the KDB files is specified by the SSLKEYR queue manager attribute.

This scenario uses the default value of SSLKEYR that is set when the queue manager is created, `/var/mqm/qmgrs/QMSA1/ssl/key`. This value is a good preferred value as it sits under the queue manager data directory and access controls can be easily applied and managed. Permissions on the `ssl` directory and all files within it should be restricted to only read and write access by the `mqm` user ID. User IDs that are members of the `mqm` group do not require access. Other users on the system do not require any access.

Additional material: All scripts for the creation and management of key repository files are contained in the `certs` sub-directory of `ch9admin`. To maintain file access controls, they must only be run by the `mqm` user ID. Key repository files for WebSphere MQ Explorer users are in Java keystore (JKS) format (file type `.jks`) and are stored in the `certs` directory, as are requests for certificates (file type `.req`) and received certificate files (file type `.cer`).

The **`vars.ksh`** script file (in the `certs` sub-directory of the additional material) contains the values of common variables that are sourced by other scripts from this file, as shown in Example 9-13. You can change these values if they do not suit the environment in which you are implementing the scenario.

Example 9-13 Common variables in certs/vars.ksh

```
# Common variables for certificate scripts

# KDB for queue manager
QMGR="QMSA1"  # Queue manager name
KEYR="/var/mqm/qmgrs/$QMGR/ssl/key"   # Default value of qmgr SSLKEYR attribute
KDB="$KEYR.kdb"
LBL="ibmwebspheremq$(echo $QMGR | tr [:upper:] [:lower:])"
DN="CN=$QMGR,OU=SA-W216 Residency,O=IBM ITSO,C=US"

# KDB for all users private certs
USRKEYR="/var/mqm/qmgrs/$QMGR/ssl/users"
USRKDB="$USRKEYR.kdb"

# Full MQ admin user
USRADM1="mqadm1"   # In lower case
LBLADM1="ibmwebspheremq$USRADM1"
DNADM1="CN=$USRADM1,OU=SA-W216 Residency,O=IBM ITSO,C=US"
PWDADM1="Roses854Green"

# Application 1 MQ admin user
USRAPP1="mqapp1"   # In lower case
```

```
LBLAPP1="ibmwebspheremq$USRAPP1"
DNAPP1="CN=$USRAPP1,OU=SA-W216 Residency,O=IBM ITSO,C=US"
PWDAPP1="Topaz716Cloud"
```

Run the script in Example 9-14 on page 113 to create a new CMS format key repository and associated files for the queue manager. This script adds the CA root signer certificate and the CA intermediate signer certificate, which prepares the way for generating the queue manager personal certificate in the next section, to complete the chain of certificates. You will need to set up your own CA or use a commercial CA provider and generate your own CA root and intermediate certificates. This task may involve more than one intermediate certificate, depending on your business requirements.

Example 9-14 Create queue manager key repository for queue manager using buildqmgrkdb.ksh

```
#!/usr/bin/ksh
# Run this script from mqm userid
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** Generate a strong password to stash"
PW="$(runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65)"

# Save on file for use by runmqakm (doesn't support -stashed option)
echo "$PW" >$KEYR.pw
chmod 600 $KEYR.pw     # mqm userid only has access

echo "*** Create empty key repository
Key Database File = $KDB
Password = $PW"

rm -f $KDB $KEYR.crl $KEYR.rdb $KEYR.sth
runmqakm -fips -keydb -create -db $KDB -pw "$PW" -type cms -stash -empty

echo "*** Add cert for root CA"
runmqakm -fips -cert -add -db $KDB -stashed -file $SRCD/wmqca1.cer -label "WMQCA1
MQ Root CA"

echo "*** Add cert for intermediate CA"
runmqakm -fips -cert -add -db $KDB -stashed -file $SRCD/wmqca2.cer -label "WMQCA2
MQ Intermediate CA"

echo "*** Show key repository files"
ls -la $KEYR.*
```

The runmqakm program is used to generate a long random password for the key database file (`key.kdb`). The password is stored in the stash file (`key.sth`) in an obscured format so that WebSphere MQ and the certificate management programs can access the key database without the original password needing to appear in plaintext on the command line. The stash file does not use strong encryption, as the password can be recovered using commonly available tools.

Sometimes, the runmqckm program needs to be used on the key database file; however, this program cannot use the stash file. The original password needs to be specified on the command line. For this reason, the plaintext password is also written to a text file (`key.pw`).

Provided that the proper file access controls are in place, this does not represent a security risk, as the stash file does not provide secure storage of the password anyway.

9.2.9 Generating the queue manager certificate and adding it to the key repository

Run the script in Example 9-15 to generate a key request file in the BASE64 text format.

Example 9-15 Request queue manager certificate using reqqmgrcert.ksh

```
#!/usr/bin/ksh
# Run this script from mqm userid
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** Generate CA signed cert request for queue manager's
*** personal certificate (with private key)"
TARGF="$SRCD/$QMGR.req"
echo "Label = $LBL
Distinguished Name = $DN
Cert request file = $TARGF"

runmqakm -fips -certreq -create -db $KDB -stashed -label $LBL -dn "$DN" -target
$TARGF -sigalg sha256 -size 2048
chmod 600 $TARGF

echo "*** List cert request file"
ls -la $TARGF
cat $TARGF
```

Use your CA certificate servers to submit the request file `QMSA1.req` and to generate a signed certificate in the BASE64 text format file `QMSA1.cer`.

Run the script in Example 9-16 to add the signed certificate into the queue manager's key repository.

Example 9-16 Add queue manager certificate using recvqmgrcert.ksh

```
#!/usr/bin/ksh
# Run this script from mqm userid
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** List certificate file"
CERF="$SRCD/$QMGR.cer"
echo "Cert file = $CERF"
ls -l $CERF
cat $CERF

echo "*** Receive CA signed cert for queue manager's
*** personal certificate (with private key)"
runmqakm -fips -cert -receive -db $KDB -stashed -file $CERF
```

The script in Example 9-17 can be run to list certificates in the queue manager's key repository. Add **-v** on the command line to display the details of all certificates.

Example 9-17 List queue manager key repository using listqmgrstore.ksh

```
#!/usr/bin/ksh
# Run this script from mqm userid
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** List certificates in queue manager key database file"
echo "KDB file = $KDB"
runmqakm -fips -cert -list all -db $KDB -stashed | tee akm.tmp

if [ "$1" = -v ]; then
  echo "*** List details of certs"
  awk -F\\t '{if(NF>1) print $2}' <akm.tmp | while read LBL ; do
    echo "** Label = $LBL"
    runmqakm -fips -cert -details -label "$LBL" -db $KDB -stashed
  done
fi

rm -f akm.tmp
```

Example 9-18 shows the output of **listqmgrstore.ksh**.

Example 9-18 Output of listqmgrstore.ksh

```
*** List certificates in queue manager key database file
KDB file = /var/mqm/qmgrs/QMSA1/ssl/key.kdb
Certificates found
* default, - personal, ! trusted
!       WMQCA1 MQ Root CA
!       WMQCA2 MQ Intermediate CA
-       ibmwebspheremqqmsa1
```

9.2.10 Creating a key repository for users

This scenario uses a key database (KDB) in CMS format to store and manage certificates for users of TLS/SSL-secured client channels. The other files for password stash, CRL, and certificate request management are also present. Files are located in the queue manager's `ssl` directory and have `users` as the file name prefix. Access permissions should be highly restricted, as on the queue manager's key repository files.

Run the script in Example 9-19 to create a new CMS format key repository and the associated files for user certificates. This script adds the CA root signer certificate and the CA intermediate signer certificate, which prepares the way for generating the users' personal certificates to complete the chains of certificates. You will need to set up your own CA or use a commercial CA provider and generate your own CA root and intermediate certificates. This task may involve more than one intermediate certificate, depending on your business requirements. In this scenario, the user's CA is the same as the queue manager's CA.

Example 9-19 Create user key repository using builduserskdb.ksh

```
#!/usr/bin/ksh
# Run this script from mqm userid
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** Generate a strong password to stash"
PW="$(runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65)"

# Save on file for use by runmqakm (doesn't support -stashed option)
echo "$PW" >$USRKEYR.pw
chmod 600 $USRKEYR.pw     # mqm userid only has access

echo "*** Create empty key repository for client users private certificates
Key Database File = $USRKDB
Password = $PW"

rm -f $USRKDB $USRKEYR.crl $USRKEYR.rdb $USRKEYR.sth
runmqakm -fips -keydb -create -db $USRKDB -pw "$PW" -type cms -stash -empty

echo "*** Add cert for root CA"
runmqakm -fips -cert -add -db $USRKDB -stashed -file $SRCD/wmqca1.cer -label
"WMQCA1 MQ Root CA"

echo "*** Add cert for intermediate CA"
runmqakm -fips -cert -add -db $USRKDB -stashed -file $SRCD/wmqca2.cer -label
"WMQCA2 MQ Intermediate CA"

echo "*** Show key repository files"
ls -la $USRKEYR.*
```

For the reasons that are described in 9.2.8, “Creating a key repository for queue manager” on page 112, the plaintext password is also written to a text file (`users.pw`).

9.2.11 Generating the user certificates and adding them to the key repository

Run the scripts in Example 9-20 and Example 9-21 to generate a key request file in BASE64 text format for the two user IDs that are used by this scenario for secure role-based WebSphere MQ administration.

Example 9-20 Request mqadm1 user certificate using reqmqadm1cert.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** Generate CA signed cert request for full administrator user $USRADM1
*** personal certificate (with private key)"
TARGF="$SRCD/$USRADM1.req"
echo "Label = $LBLADM1
Distinguished Name = $DNADM1
Cert request file = $TARGF"
```

```
runmqakm -fips -certreq -create -db $USRKDB -stashed -label $LBLADM1 -dn "$DNADM1"
-target $TARGF -sigalg sha256 -size 2048
chmod 600 $TARGF

echo "*** List cert request file"
ls -la $TARGF
cat $TARGF
```

Example 9-21 Request mqapp1 user certificate using reqmqapp1cert.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** Generate CA signed cert request for restricted administrator user
$USRAPP1
*** personal certificate (with private key)"
TARGF="$SRCD/$USRAPP1.req"
echo "Label = $LBLAPP1
Distinguished Name = $DNAPP1
Cert request file = $TARGF"

runmqakm -fips -certreq -create -db $USRKDB -stashed -label $LBLAPP1 -dn "$DNAPP1"
-target $TARGF -sigalg sha256 -size 2048
chmod 600 $TARGF

echo "*** List cert request file"
ls -la $TARGF
cat $TARGF
```

Use your CA certificate servers to submit the request files `mqadm1.req` and `mqapp1.req` to generate signed certificates in BASE64 text format files `mqadm1.cer` and `mqapp1.cer`.

Run the scripts in Example 9-22 and Example 9-23 to add the signed certificates into the users' key repository.

Example 9-22 Add mqadm1 certificate using recvmqadm1cert.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** List certificate file"
CERF="$SRCD/$USRADM1.cer"
echo "Cert file = $CERF"
ls -l $CERF
cat $CERF

echo "*** Receive CA signed cert for full MQ administrator user $USRADM1
*** personal certificate (with private key) into users key database"
runmqakm -fips -cert -receive -db $USRKDB -stashed -file $CERF
```

Example 9-23 Add mqapp1 certificate using recvmqapp1cert.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** List certificate file"
CERF="$SRCD/$USRAPP1.cer"
echo "Cert file = $CERF"
ls -l $CERF
cat $CERF

echo "*** Receive CA signed cert for application MQ administrator user $USRAPP1
*** personal certificate (with private key) into users key database"
runmqakm -fips -cert -receive -db $USRKDB -stashed -file $CERF
```

The script in Example 9-24 can be run to list certificates in the users' key repository. Add **-v** on the command line to display the details of all certificates.

Example 9-24 List users' key repository using listusersstore.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

echo "*** List certificates in users key database file"
echo "KDB file = $USRKDB"
runmqakm -fips -cert -list all -db $USRKDB -stashed | tee akm.tmp

if [ "$1" = -v ]; then
  echo "*** List details of certs"
  awk -F\\t '{if(NF>1) print $2}' <akm.tmp | while read LBL ; do
    echo "** Label = $LBL"
    runmqakm -fips -cert -details -label "$LBL" -db $USRKDB -stashed
  done
fi

rm -f akm.tmp
```

Example 9-25 shows the output of **listusersstore.ksh**.

Example 9-25 Output of listusersstore.ksh

```
*** List certificates in users key database file
KDB file = /var/mqm/qmgrs/QMSA1/ssl/users.kdb
Certificates found
* default, - personal, ! trusted
!       WMQCA1 MQ Root CA
!       WMQCA2 MQ Intermediate CA
-       ibmwebspheremqmqadm1
-       ibmwebspheremqmqapp1
```

9.2.12 Building the Java keystore files for users of WebSphere MQ Explorer

Key repository files for WebSphere MQ Explorer users are in Java keystore (JKS) format (file type `.jks`). They contain all the certificates in the CA signer chain and the users' personal certificate, including their private key. There are no password stash files or other associated files.

JKS files need to be distributed and stored securely. The user must not reveal their password. WebSphere MQ Explorer allows secure storage of the password.

Run the scripts in Example 9-26 and Example 9-27 on page 120 to build the `cl.mqadm1.jks` and `cl.mqapp1.jks` key respository files and add the certificates. A user's personal certificate is directly imported from the key repository that contains the certificates for all users.

Example 9-26 Build Java keystore for user mqadm1 using buildmqadm1jks.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

DBJKS="$SRCD/cl.$USRADM1.jks"
echo "*** Create Java Key Store for use with WebSphere MQ Explorer
Key Store file = $DBJKS
Label for userid = $LBLADM1
Password = $PWDADM1"

rm -rf $DBJKS
runmqckm -keydb -create -db $DBJKS -pw $PWDADM1 -type jks

echo "*** Add cert for root CA"
runmqckm -cert -add -db $DBJKS -pw $PWDADM1 -file $SRCD/wmqca1.cer -label "WMQCA1
MQ Root CA"

echo "*** Add cert for intermediate CA"
runmqckm -cert -add -db $DBJKS -pw $PWDADM1 -file $SRCD/wmqca2.cer -label "WMQCA2
MQ Intermediate CA"

echo "*** Import cert for client user (with private key) from
*** CMS format key repository to JKS format key repository"
USRPWD="$(cat $USRKEYR.pw)"
echo "User KDB Password = $USRPWD"

runmqckm -cert -import -db $USRKDB -pw "$USRPWD" -label "$LBLADM1" -target $DBJKS
-target_pw $PWDADM1 -target_type jks

chmod 600 $DBJKS

echo "*** Show key store file"
ls -la $DBJKS
```

Example 9-27 Build Java Keystore for user mqapp1 using buildmqapp1jks.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

DBJKS="$SRCD/cl.$USRAPP1.jks"
echo "*** Create Java Key Store for use with WebSphere MQ Explorer
Key Store file = $DBJKS
Label for userid = $LBLAPP1
Password = $PWDAPP1"

rm -rf $DBJKS
runmqckm -keydb -create -db $DBJKS -pw $PWDAPP1 -type jks

echo "*** Add cert for root CA"
runmqckm -cert -add -db $DBJKS -pw $PWDAPP1 -file $SRCD/wmqca1.cer -label "WMQCA1
MQ Root CA"

echo "*** Add cert for intermediate CA"
runmqckm -cert -add -db $DBJKS -pw $PWDAPP1 -file $SRCD/wmqca2.cer -label "WMQCA2
MQ Intermediate CA"

echo "*** Import cert for client user (with private key) from
*** CMS format key repository to JKS format key repository"
USRPWD="$(cat $USRKEYR.pw)"
echo "User KDB Password = $USRPWD"

runmqckm -cert -import -db $USRKDB -pw "$USRPWD" -label "$LBLAPP1" -target $DBJKS
-target_pw $PWDAPP1 -target_type jks

chmod 600 $DBJKS

echo "*** Show key store file"
ls -la $DBJKS
```

These two scripts generate the JKS files named `cl.mqadm1.jks` and `cl.mqapp1.jks` that are used with WebSphere MQ Explorer for two of the administration roles in this scenario.

The script in Example 9-28 can be run to list certificates in the `mqadm1` user's JKS. The **`listmqapp1jks.ksh`** script can be run to list certificates in the `mqapp1` user's JKS.

Example 9-28 List mqadm1 users JKS using listmqadm1jks.ksh

```
#!/usr/bin/ksh
# Run this script from mqm user ID
SRCD="$(dirname $0)"
. $SRCD/vars.ksh

USRKS="$SRCD/cl.$USRADM1.jks"
echo "*** List key store for a Java client
Key Store File = $USRKS
Password = $PWDADM1"

runmqckm -cert -list all -db $USRKS -pw $PWDADM1 | tee ckm.tmp
```

```
if [ "$1" = -v ]; then
  echo "*** List details of certs"
  tail -n +2 ckm.tmp | while read LBL ; do
    echo "** Label = $LBL"
    runmqckm -cert -details -label "$LBL" -db $USRKS -pw $PWDADM1
  done
fi

rm -f ckm.tmp
```

Example 9-29 shows the output of **listmqadm1jks.ksh**.

Example 9-29 Output of listmqadm1jks.ksh

```
*** List key store for a Java client
Key Store File = ./cl.mqadm1.jks
Password = Roses854Green
Certificates in database /var/mqm/ch9admin/certs/./cl.mqadm1.jks:
   ibmwebspheremqmqadm1
   wmqca1 mq root ca
   wmqca2 mq intermediate ca
```

9.2.13 Setting up the WebSphere MQ Explorer workstation

As preparation for all the administration roles demonstrated in the following sections, these steps are performed on the Windows workstation that has WebSphere MQ Explorer installed. Follow these steps:

1. Log on to Windows. The user ID should not be the same as any user ID on the queue manager server, be a Windows Administrator, or have any other special rights or group memberships in order to avoid any accidental security relationships that might affect the outcome of authentication or authorization checks in WebSphere MQ.
2. Use Windows Explorer to create a new folder to store the JKS files for all users. `C:\mqm` is preferred because this folder is the default folder in the WebSphere MQ Explorer configuration dialog windows for entering JKS file names.
3. Copy the following JKS files from the sub-directory `certs` on the `wmqsvr1` server to the new folder:
   - `cl.mqadm1.jks`
   - `cl.mqapp1.jks`
4. Start WebSphere MQ Explorer.
5. In the MQ Explorer - Navigator pane, right-click **IBM WebSphere MQ** and select **Password Store** from the pop-up menu.
6. The Password Store window opens. Click **Open Preferences**.
7. The Preferences (Filtered) window opens, as shown in Figure 9-2 on page 122.

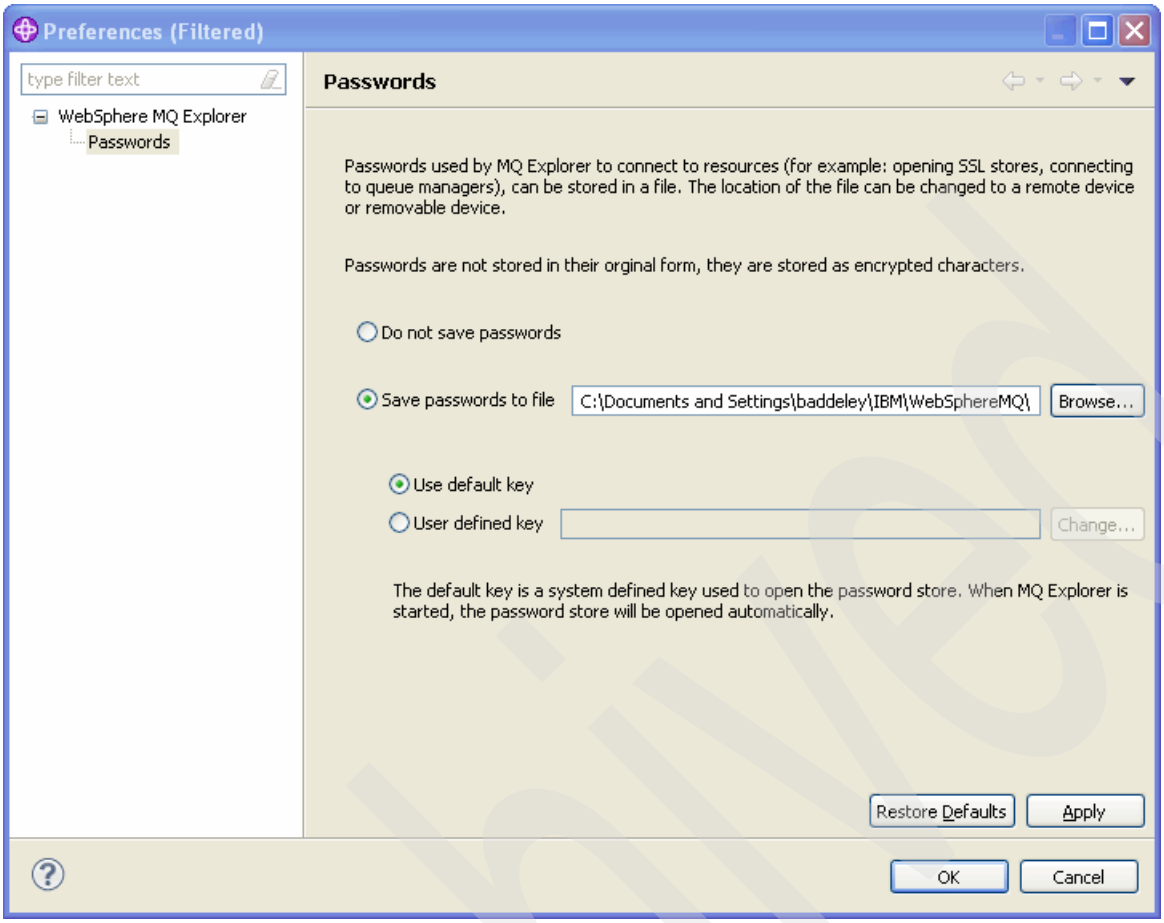


Figure 9-2 Preferences (filtered) window for passwords

Follow these steps:

– Verify that **Save passwords to file** is selected. It is not necessary to browse or change the default file name.

– Verify that **Use default key** is selected. You can select the option for User defined key and set your own key string, but this option is not necessary for the purposes of this scenario.

Click **OK** to accept any changes and close the window.

8. In the Password Store window, click **Close** to close the window, which returns you to the main WebSphere MQ Explorer window. The main window can be left open if you are now going to proceed with configuring and verify one of the administration roles.

The following sections describe the configuration of three administration roles and how they can be used. For testing, each of these roles can be implemented under the same Windows user ID by removing the queue manager in WebSphere MQ Explorer and then adding it back in again using a different configuration. It is assumed that the user is familiar with WebSphere MQ Explorer, including adding queue managers, configuring connections, and removing queue managers.

9.3 Configuring WebSphere MQ Explorer for the anonymous administration role

This section describes the configuration and use of WebSphere MQ Explorer to perform very limited administration functions on a queue manager, such as displaying objects and attributes but not being able to define new objects, change any attributes, or browse any messages.

WebSphere MQ Explorer is configured to use the default client channel `SYSTEM.ADMIN.SVRCONN`. This channel is configured on the queue manager to allow anonymous remote access from any IP address. Other administration client tools and monitors could also use this channel without any secure authentication. The use of this channel may or may not be desirable, depending on your business operation and security requirements. The authority for anonymous client users could be increased to allow messages to be browsed or other operations that do not result in any change to the queue manager.

Controls were implemented on the channel in 9.2.6, “Defining a channel for anonymous remote WebSphere MQ Explorer” on page 108 so that it runs with a fixed MCAUSER of `mqanon` for all remote connections to the channel.

The `mqanon` user ID is in the `mqanong` group, which has authority to submit PCF commands to the queue manager’s command server, but the authority of this group is restricted to only allow the display of names and attributes of the different types of objects. This authority was set up in 9.2.3, “Authorizing queue manager and system objects to enable remote WebSphere MQ Explorer” on page 104.

The following sections describe how to configure WebSphere MQ Explorer, verify that objects are displayed, and verify that attributes cannot be changed. Instructions are provided for removing the queue manager from WebSphere MQ Explorer so that other administration roles can be configured and verified.

9.3.1 Configuring a new queue manager

Follow these steps to configure the `QMSA1` queue manager in WebSphere MQ Explorer for anonymous remote administration with display-only access:

1. In the MQ Explorer - Navigator pane, right-click the **Queue Managers** folder and select **Add Remote Queue Manager** from the pop-up menu.
2. In the Add Queue Manager window, follow these steps:
   a. Enter `QMSA1` in the text box for Queue manager name. It must be in uppercase.
   b. Click the option to select **Connect directly**.
   c. Click **Next**.

3. In the Connection details section, specify the new connection:
   a. In the Host name or IP address text box, enter the host name or IP address of the queue manager. In the scenario that was developed for this book, it is `9.42.170.149`.
   b. In the Port number text box, enter the port number of the queue manager's TCP/IP listener. In the scenario that was developed for this book, it is `14151`.
   c. Confirm that the Server-connection channel text box is set to its default value of `SYSTEM.ADMIN.SVRCONN`.
   d. Clear the check box for "Automatically refresh information shown for this queue manager" as it is not necessary to dynamically show changes that are made by others.

   The window appears as shown in Figure 9-3.

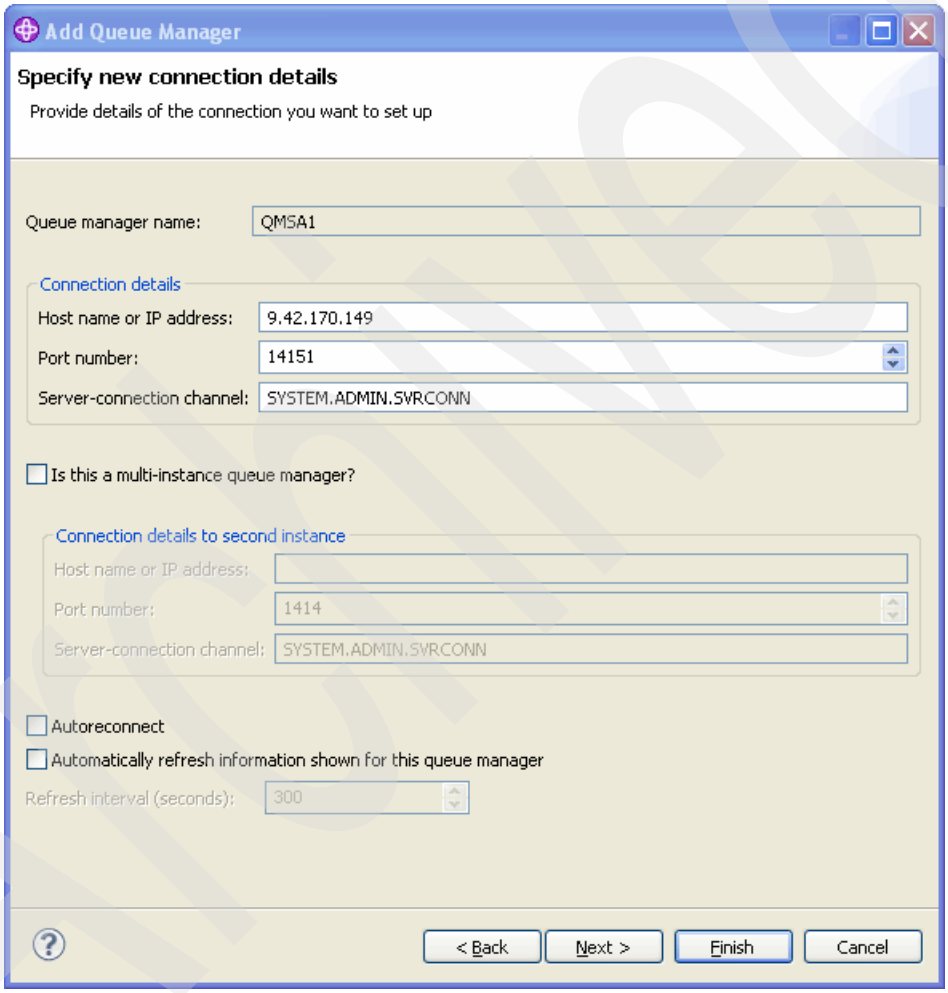


Figure 9-3 Add Queue Manager window for anonymous administration role

4. Click **Finish**.

WebSphere MQ Explorer will now attempt to connect to the queue manager. Assuming that the connection was successful, focus returns to the main WebSphere MQ Explorer window. If the connection failed, diagnose and fix the problem by reading the error message and reviewing the queue manager error logs and client error logs on the Linux system.

9.3.2 Verifying the display of objects

Follow these steps to verify that the queue manager objects can be displayed from WebSphere MQ Explorer using this configuration:

1. Click the **+** icon to the left of the queue manager name `QMSA1` to list folders for all the object types.
2. Click the **Queues** folder to display a list of nine queue names in the MQ Explorer - Content pane on the right side of the window, as shown in Figure 9-4 on page 125. This window includes object attributes that can be viewed by moving the object list to the left using the horizontal scroll bar.

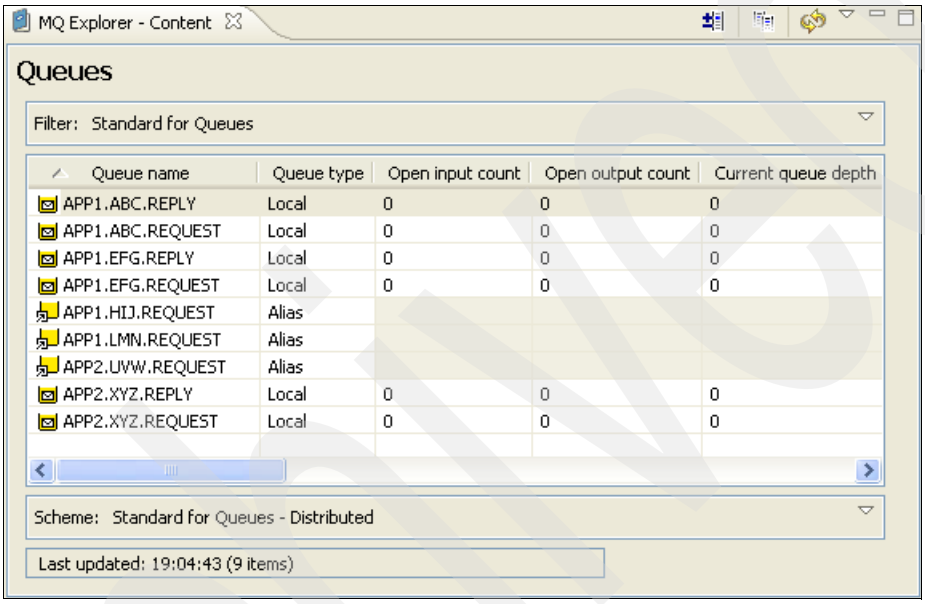


Figure 9-4 Queue objects displayed for anonymous administration role

9.3.3 Verifying that the user has no authority to alter objects

Follow these steps to demonstrate that the WebSphere MQ Explorer user only has authority to display objects:

1. Double-click the first queue, **APP1.ABC.REPLY** (in Figure 9-4), to open the Properties window for the object.
2. Click the pull-down arrow for the Put messages attribute.
3. In the display list of valid attribute values, click **Inhibited**. The value of the attribute is now highlighted and changes from `Enabled` to `Inhibited`.
4. To request the queue manager to change the value of this attribute, click **Apply**.

   This step should fail, as indicated by an error window that appears as shown in Figure 9-5.

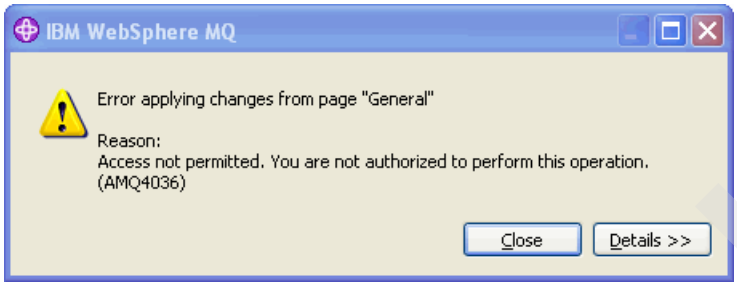


Figure 9-5 Access not permitted to alter object for anonymous administration role

5. Click **Close** to close the `error` window.
6. Click **Cancel** to close the Properties window.

You can try to alter any attribute of any other object, including the queue manager object. It should always fail with a reason of `Access not permitted. You are not authorized to perform this operation (AMQ4036)`.

This point concludes the demonstration of anonymous administration in this scenario.

9.3.4 Removing the queue manager from WebSphere MQ Explorer

To remove the queue manager from WebSphere MQ Explorer so that other administration roles in this scenario can be configured and verified, follow these steps:

1. In the MQ Explorer - Navigator pane, right-click the **QMSA1** queue manager folder and select **Disconnect** from the pop-up menu. In the MQ Explorer - Content pane, the Connection status should be shown as `Disconnected`.
2. Right-click the **QMSA1** queue manager folder again and select **Remove** from the pop-up menu.
3. In the new window that asks `Remove the queue manager 'QMSA1' (AMQ4133)`, click **Yes**.

Focus returns to the main WebSphere MQ Explorer window and the queue manager is no longer listed in the Queue Managers folder.

9.4 Configuring WebSphere MQ Explorer for a limited administration role

This section describes the configuration and use of WebSphere MQ Explorer to perform limited administration functions on a queue manager, such as displaying objects and attributes and being able to change attributes and browse messages. New objects cannot be defined. This role might be used for application operations or support.

In this scenario, the strongly authenticated administrator user `mqapp1` is authorized to perform some administration functions for object names that begin with `APP1` but only has display authority for other objects, including object names that begin with `APP2`.

WebSphere MQ Explorer is configured to use client channel `ADMIN.QMSA1`. It is configured on the queue manager to use strong TLS/SSL two-way authentication and encryption of messages over the channel.

The following controls were implemented on the channel in 9.2.7, “Defining a secure channel for remote administration roles” on page 109:

- The channel runs with TLS/SSL security and client authentication is required.
- Part of the client’s DN “`OU=SA-W216 Residency,O=IBM ITSO,C=US`” is filtered on the channel SSLPEER attribute.
- The complete DN “`CN=mqapp1,OU=SA-W216 Residency,O=IBM ITSO,C=US`” is filtered on a CHLAUTH SSLPEERMAP rule to set the MCAUSER to `mqapp1`.
- The client’s IP address is also filtered on “`9.42.*`” using the same CHLAUTH rule.

The `mqapp1` user ID is in the `mqappg1` group, which has authority to submit PCF commands to the queue manager’s command server. The authority of this group allows the display of names of the different types of objects and their attributes. This authority is the same as the anonymous user in the previous section. The group has additional administration authorities (`get`, `browse`, `put`, `inq`, `set`, `chg`, and `clr` settings) on queue object names that begin with `APP1.` and “`inq`, `set`, and `chg`” authorities on process object names that begin with `APP1.`, as set up in 9.2.4, “Defining application objects and limited administration authority” on page 105.

The following sections describe how to configure WebSphere MQ Explorer and verify that attributes can be changed on APP1 queues and process objects. It is also verified that queue and process objects cannot be changed on other objects.

9.4.1 Configuring the new queue manager

Follow these steps to configure the `QMSA1` queue manager in WebSphere MQ Explorer for remote access with limited administration authority to change some objects:

1. In the MQ Explorer - Navigator pane, right-click the **Queue Managers** folder and select **Add Remote Queue Manager** from the pop-up menu.
2. In the Add Queue Manager window, follow these steps:
   a. Enter `QMSA1` in the text box for Queue manager name. It must be in uppercase.
   b. Confirm that the default option is selected to **Connect directly**.
   c. Click **Next**.
3. In the Connection details section, specify the new connection:
   a. In the Host name or IP address text box, enter the host name or IP address of the queue manager. In the scenario that was developed for this book, it is `9.42.170.149`.
   b. In the Port number text box, enter the port number of the queue manager’s TCP/IP listener. In the scenario that was developed for this book, it is `14151`.
   c. In the Server-connection channel text box, enter `ADMIN.QMSA1`.
   d. Clear the check box for “Automatically refresh information shown for this queue manager” as it is not necessary to dynamically show changes that are made by others.

   The window now appears as shown in Figure 9-6 on page 128.

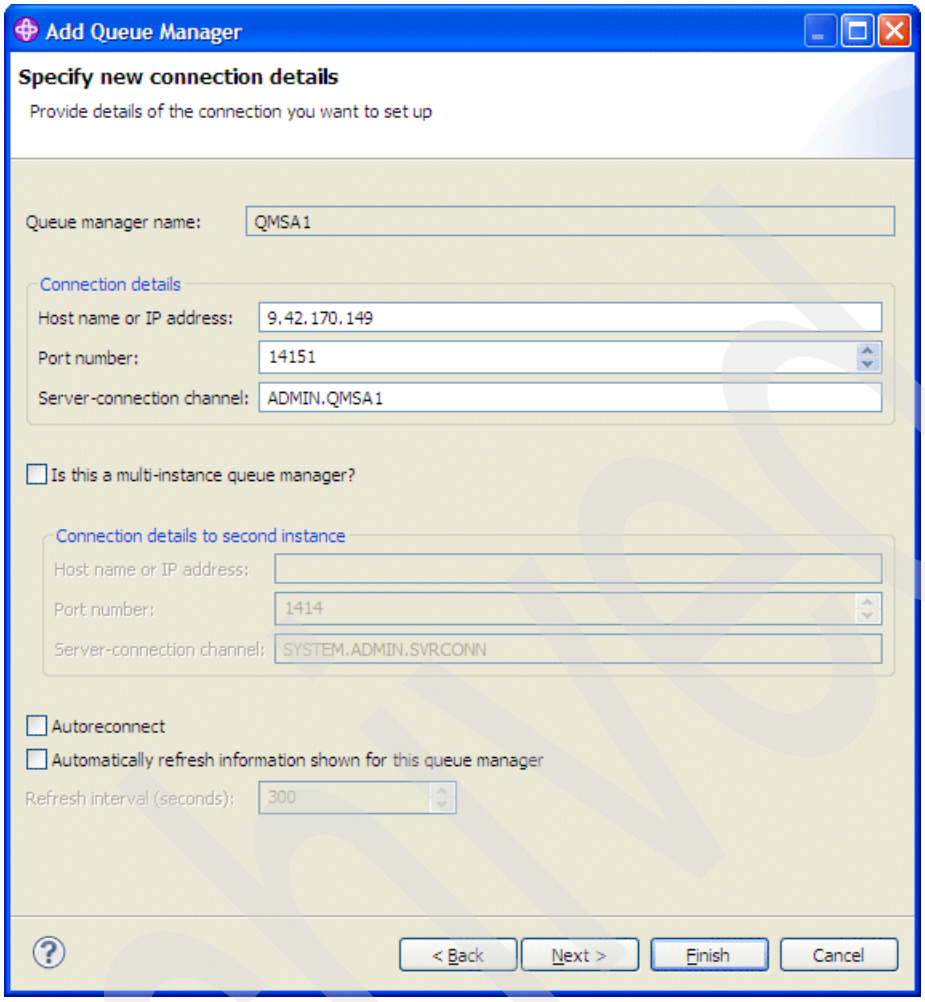


Figure 9-6 Add Queue Manager window for limited administration role

4. Click **Next** twice to skip over the security exit details pane and user identification details pane of the Add Queue Manager window.
5. The next pane configures the locations and passwords of the SSL certificate key repositories. Click **Enable SSL key repositories** to enable this capability.
6. In the upper section, Trusted Certificate Store, click **Browse**. The Open window opens and lists the file names of type "`.jks`" that are in the `C:\mqm` directory. Double-click file name **cl.mqapp1.jks** in the list and the window will close.
7. In the Add Queue Manager window section Trusted Certificate Store, click **Enter password**. This selection displays the Password details text dialog window. Enter the password of the Java Keystore file `cl.mqapp1.jks`, as originally specified in script file **`certs/vars.ksh`** in 9.2.8, "Creating a key repository for queue manager" on page 112 by the value of `PWDAPP1` variable. Click **OK** to close the text dialog window.
8. In the Add Queue Manager window section Personal Certificate Store, click **Browse**. The Open window opens and lists the file names of type "`.jks`" that are in the `C:\mqm` directory. Double-click file name **cl.mqapp1.jks** in the list and the window will close. One JKS file is used for both the trusted and personal certificate stores.
9. In the Add Queue Manager window section Personal Certificate Store, click **Enter password**. This selection displays the Password details text dialog window. Enter the same password. Click **OK** to close the text dialog window.

The Add Queue Manager window now appears as shown in Figure 9-7.

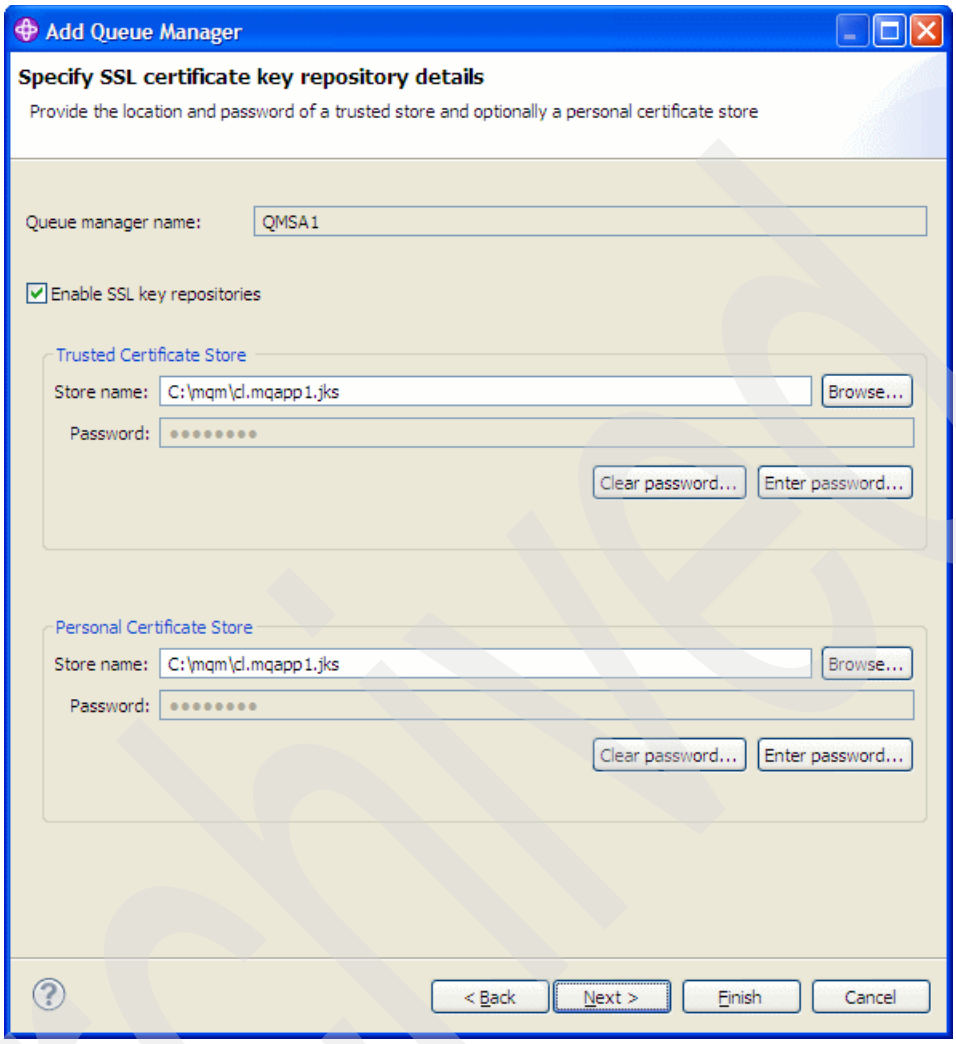


Figure 9-7 SSL certificate key repository details for limited administration role

10. Click **Next** to show the SSL configuration options. Follow these steps:
    a. Click the check box for **Enable SSL options** to enable the options.
    b. On the SSL CipherSpec list, click the down arrow and click the entry near the bottom of the list for **TLS_RSA_WITH_AES_128_CBC_SHA**.

    The Add Queue Manager window now appears as shown in Figure 9-8 on page 130.

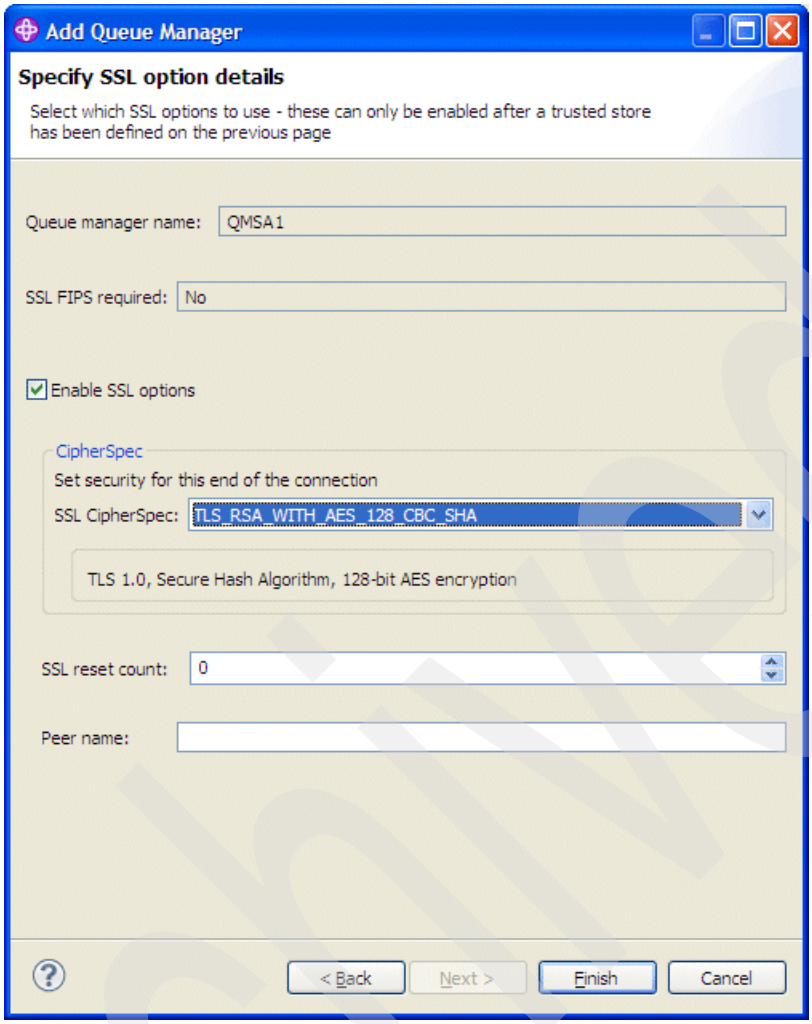


Figure 9-8 Specify SSL option details for limited administration role

11. Click **Finish**.

WebSphere MQ Explorer will now attempt to connect to the queue manager. Assuming that the connection was successful, focus returns to the main WebSphere MQ Explorer window. If the connection failed, diagnose and fix the problem by reading the error message, and look at the queue manager error logs and client error logs on the Linux system.

9.4.2 Displaying the channel status

The use of TLS/SSL security and client authentication can be verified by displaying the status of the running channel on the queue manager.

Run the script that is shown in Example 9-30 on the queue manager system (`dischs.ksh` in the `ch9admin` directory of the additional materials) to display the channels.

Example 9-30 Displaying the channels

```
#!/usr/bin/ksh
. ./vars.ksh
echo "dis chs(*) all" | runmqsc $QMGR
```

The output of the script should be very similar to Example 9-31. Verify that MCAUSER is `mqapp1` and that the CN field in SSLPEER is also `mqapp1`.

Example 9-31 Channel status for limited administration role

```
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMSA1.
1 : dis chs(*) all
AMQ8417: Display Channel Status details.
   CHANNEL(ADMIN.QMSA1)                    CHLTYPE(SVRCONN)
   BUFSRCVD(142)                           BUFSSENT(152)
   BYTSRCVD(36384)                         BYTSSENT(90796)
   CHSTADA(2012-07-18)                     CHSTATI(10.46.48)
   COMPHDR(NONE,NONE)                      COMPMSG(NONE,NONE)
   COMPRATE(0,0)                           COMPTIME(0,0)
   CONNAME(9.42.171.189)                   CURRENT
   EXITTIME(0,0)                           HBINT(300)
   JOBNAME(00003F0F00000007)               LOCLADDR(::ffff:9.42.170.149(14151))
   LSTMSGDA(2012-07-18)                    LSTMSGTI(10.53.50)
   MCASTAT(RUNNING)                        MCAUSER(mqapp1)
   MONCHL(OFF)                             MSGS(151)
   RAPPLTAG(WebSphere MQ Client for Java)
   SSLCERTI(CN=WMQCA2 MQ Intermediate CA,OU=SA-W216 Residency,O=IBM ITSO,C=US)
   SSLKEYDA( )                             SSLKEYTI( )
   SSLPEER(SERIALNUMBER=52:11:10:44:00:00:00:00:00:42,CN=mqapp1,OU=SA-W216
Residency,O=IBM ITSO,C=US)
   SSLRKEYS(0)                             STATUS(RUNNING)
   STOPREQ(NO)                             SUBSTATE(RECEIVE)
   CURSHCNV(1)                             MAXSHCNV(10)
   RVERSION(00000000)                      RPRODUCT(MQJB)
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

9.4.3 Verifying that the user can display objects

Use the following steps to verify that the WebSphere MQ Explorer user can display objects:

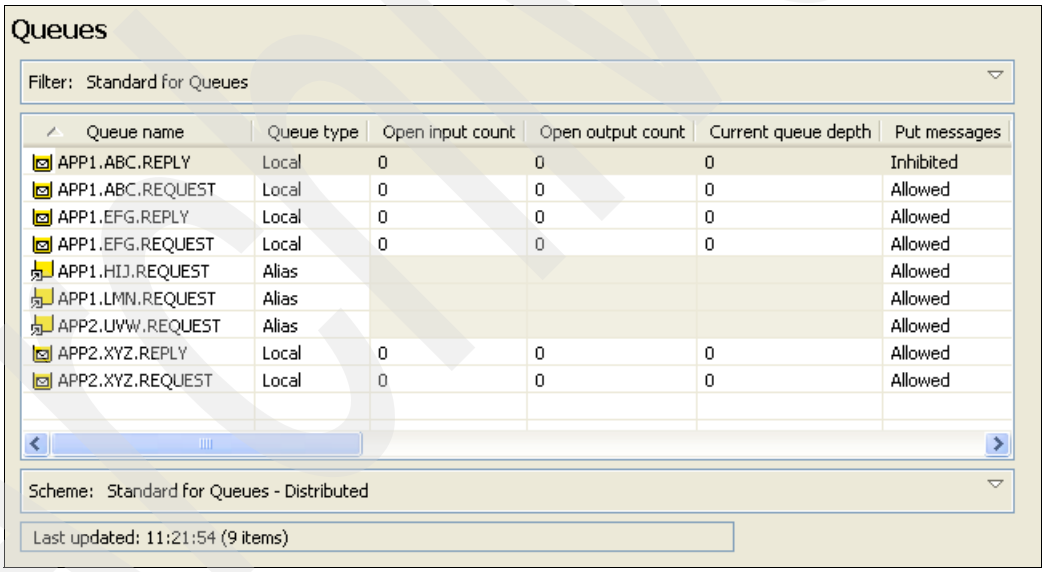
1. Click the + icon to the left of the queue manager name `QMSA1` to list folders for all the object types.
2. Click the **Queues** folder. This selection displays a list of nine queue names in the MQ Explorer - Content pane on the right side of the window, which is exactly the same as shown in Figure 9-4 on page 125 for the anonymous administration user. This list includes object attributes that can be viewed by moving the object list to the left by using the horizontal scroll bar. A limited administration role should have at least the same capabilities as the anonymous administration role.

9.4.4 Verifying that the user has authority to alter APP1 objects

Follow these steps to demonstrate that the WebSphere MQ Explorer user can alter one of their objects:

1. Double-click the first queue name **APP1.ABC.REPLY** in the display from the previous section to open the Properties window for the object.
2. Click the pull-down arrow for the Put messages attribute.
3. In the display of the list of valid attribute values, click **Inhibited**. The value of the attribute is now highlighted and changes from `Enabled` to `Inhibited`.
4. Click **OK** to request the queue manager to change the value of this attribute and close the window.
5. The focus returns to the main WebSphere MQ Explorer window. A progress bar should become active for a short time and then the displayed list of queues and attributes will be refreshed.
6. Make the WebSphere MQ Explorer window larger by moving the mouse to the right border. The mouse pointer will change to left-right arrows. Hold the left mouse button down and drag the border to the right to make the window wider so that the `Put Messages` column is visible in the window.

   Figure 9-9 shows that the `Put Messages` attribute has been successfully changed to `Inhibited` on the `APP1.ABC.REPLY` queue.



Queue name	Queue type	Open input count	Open output count	Current queue depth	Put messages
APP1.ABC.REPLY	Local	0	0	0	Inhibited
APP1.ABC.REQUEST	Local	0	0	0	Allowed
APP1.EFG.REPLY	Local	0	0	0	Allowed
APP1.EFG.REQUEST	Local	0	0	0	Allowed
APP1.HIJ.REQUEST	Alias				Allowed
APP1.LMN.REQUEST	Alias				Allowed
APP2.UVW.REQUEST	Alias				Allowed
APP2.XYZ.REPLY	Local	0	0	0	Allowed
APP2.XYZ.REQUEST	Local	0	0	0	Allowed

Figure 9-9 Queue object changed in limited administration role

9.4.5 Verifying that the user has no authority to alter APP2 objects

Follow these steps to demonstrate that the user only has authority to display another application's objects:

1. In the list of queues (Figure 9-9), double-click the queue name **APP2.UVW.REQUEST** to open the Properties window for the alias queue object.
2. Click the pull-down arrow for the Put messages attribute.
3. In the display list of valid attribute values, click **Inhibited**. The value of the attribute is now highlighted and has changed from `Enabled` to `Inhibited`.

4. To request the queue manager to change the value of this attribute, click **Apply**.

   This request should fail, as indicated by an error window that appears as shown in Figure 9-10 on page 133.

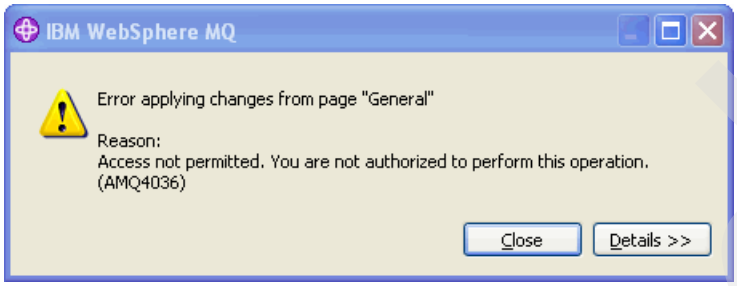


Figure 9-10 Access not permitted to alter object for limited administration role

5. Click **Close** to close the error window.
6. Click **Cancel** to close the Properties window.

On the queue manager system, you can display the last message in the error log to see the exact reason for the authorization failure. Enter the shell command:

```
tail -20 /var/mqm/qmgrs/QMSA1/errors/AMQERR01.LOG
```

Example 9-32 shows the relevant message.

Example 9-32 Queue manager error log entry for authorization failure

```
07/18/2012 11:27:12 AM - Process(14348.7) User(baddeley) Program(amqzlaa0)
                    Host(wmqsvr1.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(QMSA1)
AMQ8077: Entity 'mqapp1      ' has insufficient authority to access object
'APP2.UVW.REQUEST'.
EXPLANATION:
The specified entity is not authorized to access the required object. The
following requested permissions are unauthorized: chg
ACTION:
Ensure that the correct level of authority has been set for this entity against
the required object, or ensure that the entity is a member of a privileged
group.
----- amqzfubx.c : 624 -------------------------------------------------------
```

You can try to alter any attribute of any other `APP2` object, including the queue manager object. It should always fail with a reason of `Access not permitted. You are not authorized to perform this operation (AMQ4036)`.

This demonstration of secured limited administration in this scenario is complete.

To revert to the WebSphere MQ Explorer configuration so that another administration role can be configured and verified, see 9.3.4, “Removing the queue manager from WebSphere MQ Explorer” on page 126.

9.5 Configuring WebSphere MQ Explorer for a full administration role

This section describes the configuration and use of WebSphere MQ Explorer to perform a full administration role on a queue manager, such as displaying and changing all objects and attributes. Queued messages can be manipulated. New objects can be defined. Authority profiles can be changed and created. This role is functionally equivalent to being the `mqm` user ID on a UNIX or Linux queue manager system.

In this scenario, the strongly authenticated user `mqadm1` is authorized to perform a full administration role by virtue of being a member of the `mqm` group on the queue manager system. This type of authority and group membership should be tightly controlled. It allows everything to be changed in the queue manager. The full administrator can browse, put, and get messages on all application queues. They can possibly access to other queue managers via distributed channels or clusters.

WebSphere MQ Explorer is configured to use client channel `ADMIN.QMSA1`. It is configured on the queue manager to use strong TLS/SSL two-way authentication and encryption of messages over the channel.

The following controls were implemented on the channel in 9.2.7, “Defining a secure channel for remote administration roles” on page 109:

- The channel runs with TLS/SSL security and client authentication is required.
- Part of the client’s distinguished name (DN) “`OU=SA-W216 Residency,O=IBM ITSO,C=US`” is filtered on the channel SSLPEER attribute.
- The complete DN “`CN=mqadm1,OU=SA-W216 Residency,O=IBM ITSO,C=US`” is filtered on a CHLAUTH SSLPEERMAP rule to set the MCAUSER to `mqapp1`.
- The client’s IP address is also filtered on “`9.42.*`” using the same CHLAUTH rule.

The `mqadm1` user ID has primary and secondary membership of the `mqm` group. This group always has full authority to submit PCF commands to the queue manager’s command server.

The following sections describe how to configure WebSphere MQ Explorer and verify that attributes can be changed on all object types and the queue manager object.

Configuring a new queue manager

Follow these steps to configure the `QMSA1` queue manager in WebSphere MQ Explorer for remote access with full administration authority to all objects:

1. In the MQ Explorer - Navigator pane, right-click the **Queue Managers** folder and select **Add Remote Queue Manager** from the pop-up menu.
2. In the Add Queue Manager window, follow these steps:
   a. Enter `QMSA1` in the text box for Queue manager name. It must be in uppercase.
   b. Confirm that the default option **Connect directly** is selected.
   c. Click **Next**.

3. In the Connection details window, specify the new connection:
   a. In the Host name or IP address text box, enter the host name or IP address of the queue manager. In the scenario that was developed for this book, it is `9.42.170.149`.
   b. In the Port number text box, enter the port number of the queue manager's TCP/IP listener. In the scenario that was developed for this book, it is `14151`.
   c. In the Server-connection channel text box, enter `ADMIN.QMSA1`.
   d. Clear the check box for "Automatically refresh information shown for this queue manager" as it is not necessary to dynamically show changes that are made by others.

   The window now appears as shown in Figure 9-11 on page 135.

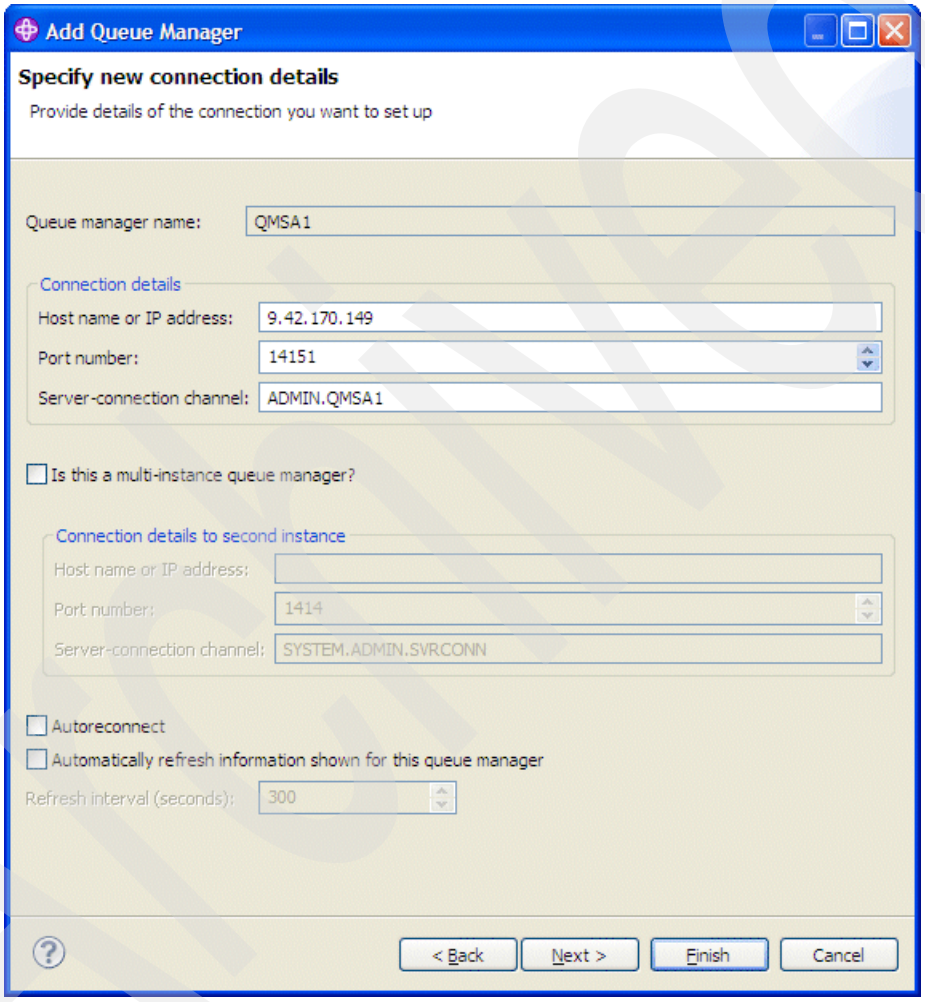


Figure 9-11 Add Queue Manager window for limited administration role

4. Click **Next** twice to skip over the security exit details pane and user identification details pane of the Add Queue Manager window. The next pane configures the locations and passwords of the SSL certificate key repositories. Follow these steps:
   a. Click the check box for **Enable SSL key repositories** to enable this capability.
   b. In the upper section, Trusted Certificate Store, click **Browse**. The Open window opens and lists the file names of type "`.jks`" that are in the `C:\mqm` directory. Double-click file name **cl.mqadm1.jks** in the list and the window will close.

c. The focus returns to the Add Queue Manager window. In the upper section, Trusted Certificate Store, click **Enter password**. This selection displays the Password details text dialog window. Enter the password of the Java keystore file `cl.mqadm1.jks`. The password was specified as the value of the *PWDADM1* variable in the **`certs/vars.ksh`** script file in 9.2.8, “Creating a key repository for queue manager” on page 112. Click **OK** to close the text dialog window.

d. The focus returns to the Add Queue Manager window. In the lower section, Personal Certificate Store, click **Browse**. The Open window opens and lists the file names of type “`.jks`” that are in the `C:\mqm` directory. Double-click the file name **cl.mqadm1.jks** in the list and the window will close. One JKS file is used for both the trusted and personal certificate stores.

e. The focus returns to the Add Queue Manager window. In the lower section, Personal Certificate Store, click **Enter password**. This option displays the Password details text dialog window. Enter the same password. Click **OK** to close the text dialog window.

The Add Queue Manager window now appears as shown in Figure 9-12.

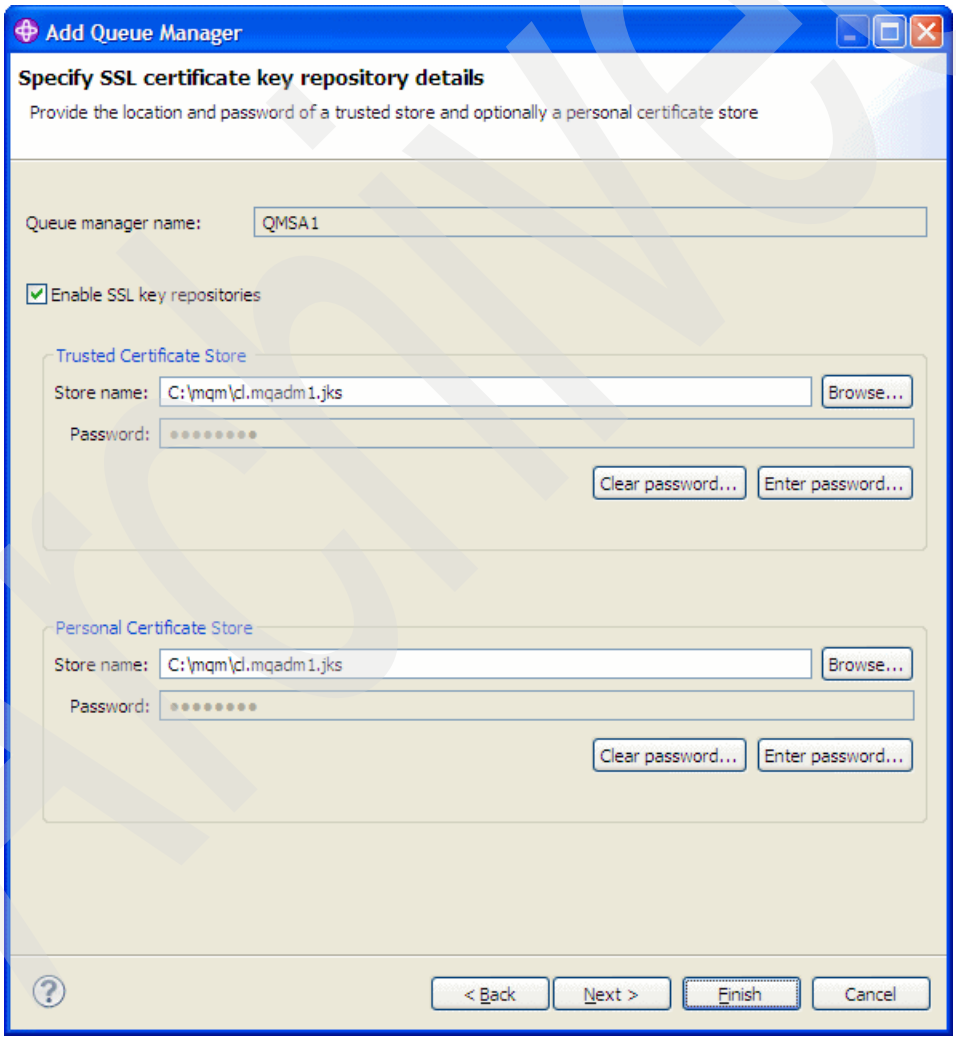


Figure 9-12 SSL certificate key repository details for full administration role

5. Click **Next** to configure the SSL options. Follow these steps:
   a. Click the check box for **Enable SSL options** to enable the SSL options.
   b. On the SSL CipherSpec list, click the down arrow and click the entry near the bottom of the list for **TLS_RSA_WITH_AES_128_CBC_SHA**.

   The Add Queue Manager window now appears as shown in Figure 9-13 on page 137.

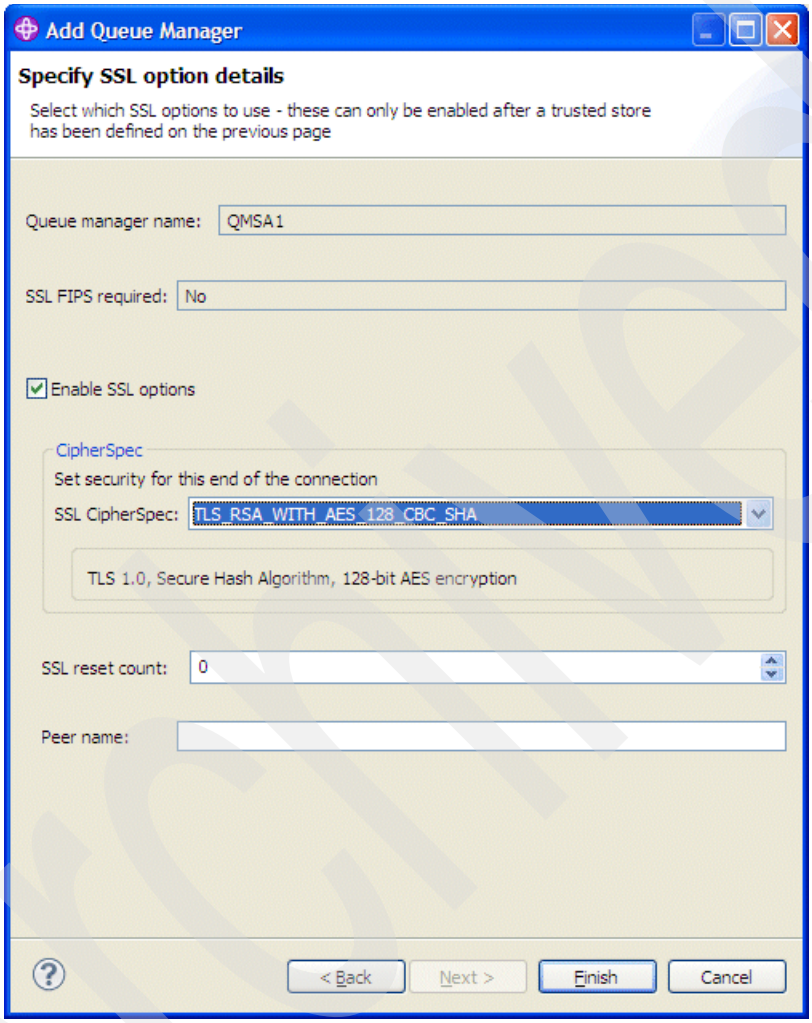


Figure 9-13 SSL options details for limited administration role

6. Click **Finish**.

WebSphere MQ Explorer will now attempt to connect to the queue manager. Assuming that the connection was successful, the focus returns to the main WebSphere MQ Explorer window. If the connection failed, diagnose and fix the problem by reading the error message and look at the queue manager error logs and client error logs on the Linux system.

9.5.1 Displaying the channel status

The use of TLS/SSL security and client authentication can be verified by displaying the status of the running channel on the queue manager.

Run the **dischs.ksh** script (Example 9-30 on page 130) to display the channels. The output of the script should be very similar to Example 9-33. Verify that MCAUSER is mqadm1 and that the CN field in SSLPEER is also mqadm1.

Example 9-33 Channel status for the full administration role

```
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMSA1.
1 : dis chs(*) all
AMQ8417: Display Channel Status details.
   CHANNEL(ADMIN.QMSA1)                    CHLTYPE(SVRCONN)
   BUFSRCVD(66)                            BUFSSENT(176)
   BYTSRCVD(17500)                         BYTSSENT(195496)
   CHSTADA(2012-07-18)                     CHSTATI(13.22.42)
   COMPHDR(NONE,NONE)                      COMPMSG(NONE,NONE)
   COMPRATE(0,0)                           COMPTIME(0,0)
   CONNAME(9.42.171.189)                   CURRENT
   EXITTIME(0,0)                           HBINT(300)
   JOBNAME(00003F0F0000000B)               LOCLADDR(::ffff:9.42.170.149(14151))
   LSTMSGDA(2012-07-18)                    LSTMSGTI(13.22.46)
   MCASTAT(RUNNING)                        MCAUSER(mqadm1)
   MONCHL(OFF)                             MSGS(175)
   RAPPLTAG(WebSphere MQ Client for Java)
   SSLCERTI(CN=WMQCA2 MQ Intermediate CA,OU=SA-W216 Residency,O=IBM ITSO,C=US)
   SSLKEYDA( )                             SSLKEYTI( )
   SSLPEER(SERIALNUMBER=52:0F:2C:F8:00:00:00:00:00:41,CN=mqadm1,OU=SA-W216
Residency,O=IBM ITSO,C=US)
   SSLRKEYS(0)                             STATUS(RUNNING)
   STOPREQ(NO)                             SUBSTATE(RECEIVE)
   CURSHCNV(1)                             MAXSHCNV(10)
   RVERSION(00000000)                      RPRODUCT(MQJB)
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

Verifying the authority to alter any object

To verify that the WebSphere MQ Explorer user can alter objects, follow these steps:

1. Click the + icon to the left of the queue manager name QMSA1 to list folders for all the object types.
2. Click the **Queues** folder. This selection displays a list of nine queue names in the MQ Explorer - Content pane on the right side of the window, which is exactly the same as shown in Figure 9-4 on page 125 for the anonymous administration user, and also for the limited administration user.
3. Click the +- icon near the upper-right corner of the pane. The list is refreshed to add all the SYSTEM objects to the list of queue names.

   Figure 9-14 on page 139 shows the list of all queue objects, including SYSTEM queues.

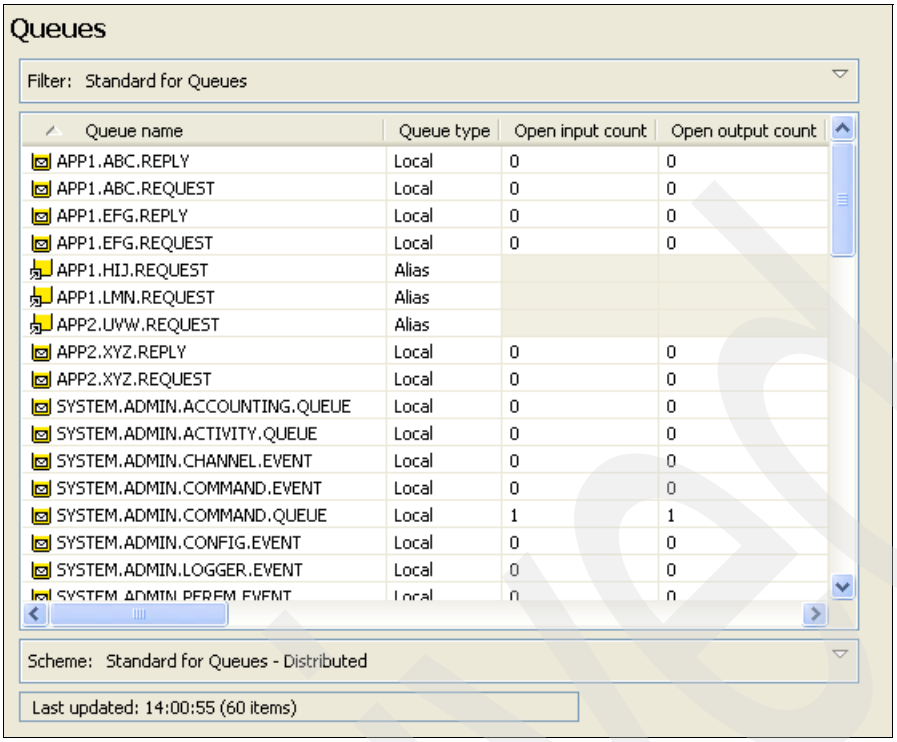


Figure 9-14 Queue objects displayed for full administrator role

4. One of the SYSTEM queues is used to demonstrate that the user can alter any object. Double-click the first SYSTEM queue object named **SYSTEM.ADMIN.ACCOUNTING.QUEUE** to open the Properties window for the object.
5. Click the pull-down arrow for the Get messages attribute.
6. In the display list of valid attribute values, click **Inhibited**. The value of the attribute is now highlighted and changes from `Enabled` to `Inhibited`.
7. To request the queue manager to change the value of this attribute and close the window, click **OK**. The focus returns to the main WebSphere MQ Explorer window. A progress bar should become active for a short time and then the displayed list of queues and attributes will be refreshed.
8. Make the WebSphere MQ Explorer window larger by moving the mouse to the right border. The mouse pointer will change to left-right arrows. Hold the left mouse button down and drag the border to the right to make the window wider, so that the `Put Messages` column is visible in the window.

   Figure 9-15 on page 140 shows that the `Get Messages` attribute has been successfully changed to `Inhibited` on the `SYSTEM.ADMIN.ACCOUNTING.QUEUE` queue.

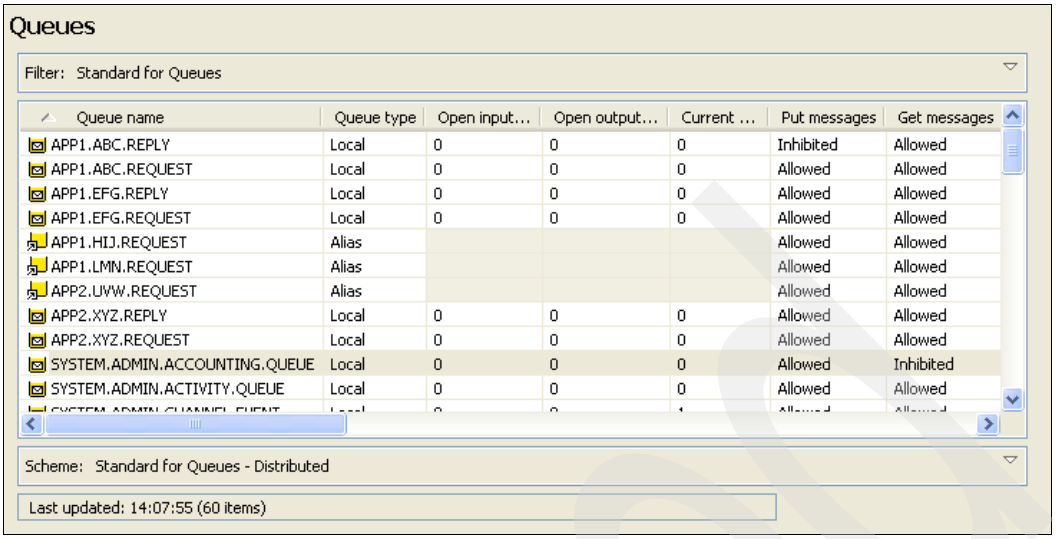


Figure 9-15 Queue object changed in full administration role

You can also try changing an attribute of the queue manager object by right-clicking the queue manager icon and selecting **Properties** from the pop-up menu. Change the Description field to any text value and click **OK**. The MQ Explorer - Content pane will be updated to show the new value of Description down near the bottom of the pane.

This demonstration of secured full administration in this scenario is complete.

To revert to the WebSphere MQ Explorer configuration so that another administration role can be tried, see 9.3.4, “Removing the queue manager from WebSphere MQ Explorer” on page 126.

9.6 Summary

There are many possible roles for administration of WebSphere MQ queue managers. This scenario demonstrated the configuration and usage of three roles. WebSphere MQ Explorer running on a Windows workstation was used by the roles to connect to a queue manager on Linux.

The three roles represented a broad spectrum, from anonymous unsecured access through to highly secured full administrator access. Highly secured limited administrator access was also demonstrated as a possible intermediate role between these two roles.

The anonymous role used a well-known channel name with few security controls. All remote clients run under the authority of the same user ID that only has authority to display object names and attributes.

The two other roles used strongly authenticated connections on a dedicated channel. To further improve security, the name of this channel should only be known by those individuals who need to use it.

Appropriate authorization controls were placed on the objects to be administered. Positive and negative verification tests were performed.

10

Scenario: Securing IBM WebSphere MQ connections to connect a business partner

This chapter presents a scenario that connects a business application to a business partner. The application runs on a server and uses an IBM WebSphere MQ client to connect to an application queue manager. Connectivity to the application queue manager uses Secure Sockets Layer (SSL)/Transport Layer Security (TLS) for authentication and encryption, and WebSphere MQ object authority manager (OAM) for authorization. Connections between queue managers also use SSL/TLS and OAM for the authentication, authorization, and privacy. This scenario is a typical WebSphere MQ configuration for an environment where privacy and access control are important. The prerequisites and setup steps for all scenarios in this book can be found in Chapter 8, “Scenario preparation” on page 83.

This chapter contains the following topics:

- Scenario overview
- WebSphere MQ and WebSphere MQ Internet pass-thru features and practices
- Implementing the scenario
- Results of testing
- Advanced Encryption Standard support in WebSphere MQ Internet pass-thru
- Summary

10.1 Scenario overview

In a business-to-business (B2B) interface, the goal is to build a safe connection between two separate administrative domains. The local administrator has little control over the security design or process rigor employed by the business partner and so is unable to control the risk to the same degree. The result is that the security design for B2B generally applies a greater number of controls, more stringent controls, or both.

For example, a B2B interface with multiple business partners must not only protect the internal systems from a rogue partner, but they must also isolate the partners from one another. Or the network may allow anonymous inquire access for internal users while requiring only authenticated and restricted access from business partners. Additional security controls and increased complexity have an associated cost in performance and administrative overhead so the B2B security design would not be appropriate to use across the entire network.

The techniques that are presented in this chapter are more complex than the techniques that are presented in Chapter 6, “WebSphere MQ security controls” on page 41, but at the same time, it is a compromise of cost versus benefit. There are considerably more security controls available than are demonstrated here but to implement all of them would exceed most real-world requirements and incur a cost that might very well exceed the business benefit expected from the interface. The best designs will balance cost against benefit and deliberately choose security controls that meet specific objectives. The design shown is one based on real-world implementations that require strong authentication, granular authorization, and isolation of business partners from negatively impacting the internal systems or one another.

10.1.1 Scenario design

The scenario illustrates a connection between two companies, Company1 and Company2. The scenario is based on the viewpoint of Company1. The WebSphere MQ administrator in Company2 does not have access to the configuration of Company1’s network and this design is as it should be. The only things that are visible to the business partner are the IP address, the channel name, the certificate details (signer chain and distinguished name (DN)) and the (possibly aliased) queue manager name of the gateway queue manager. All other details of Company1’s internal WebSphere MQ configuration, such as the names of the queue managers, are hidden from the business partners.

The scenario describes how a message can flow securely from the application queue manager, through a gateway and proxy, to the partner. This scenario is clearly more complex than a normal connection between two queue managers. This complex connection path meets security guidelines, including terminating SSL/TLS sessions in the outer DMZ, and protecting application queue managers by having the gateway queue manager between the application and the business partner. Figure 10-1 on page 143 shows the connectivity between the application and the business partner.

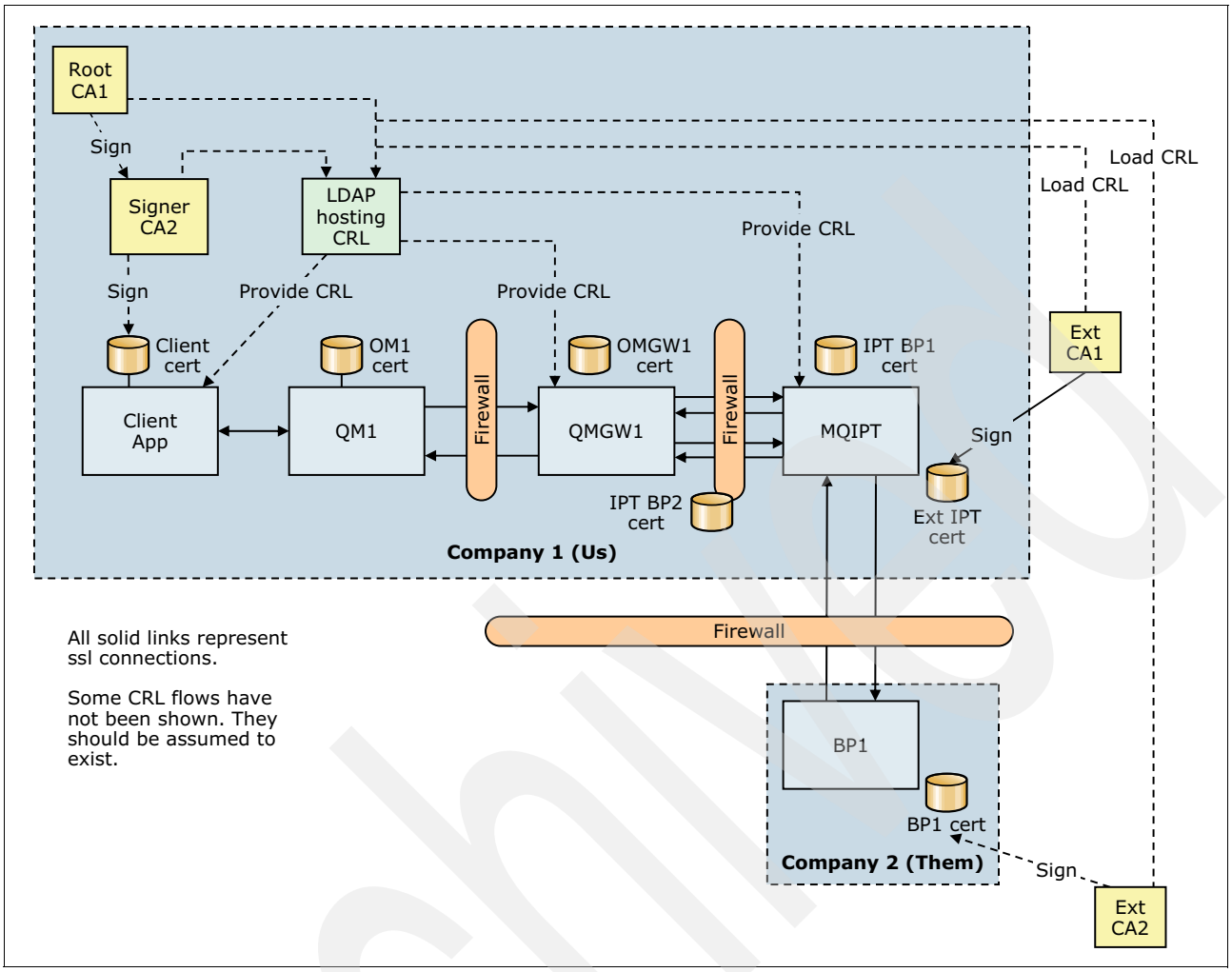


Figure 10-1 Application to business partner connectivity, including certificate stores and signing

In this topology, WebSphere MQ Internet pass-thru (SupportPac MS81) is used on a dedicated server in the partner-facing DMZ, where it acts as a secure proxy for SSL connections. A proxy, rather than a queue manager, is used to terminate the SSL in the external DMZ. A queue manager in the DMZ would write unencrypted persistent messages onto disk. Using WebSphere MQ Internet pass-thru as a proxy service in the DMZ reduces the risk of unencrypted message data being stored to disk in the DMZ. The WebSphere MQ Internet pass-thru service also has better separation of trust settings than is possible with WebSphere MQ alone.

The scenario describes a security practice of *defense in depth*; even if the attacker finds a way through one layer of protection, there is another protection to stop the attack before significant damage can be done.

The scenario illustrates the following security guidelines and techniques:

- Terminating SSL/TLS connections in the external DMZ
- Authenticating business partner certificates as strongly as possible
- Controlling and limiting business partner access to destinations (queues/queue managers)
- Hiding the specifics of internal queue manager topology (including names)

- Using firewalls to limit attack options available to attackers
- Using multiple mechanisms to control access (IP *whitelist*, SSL certificates, and firewalls)

In addition, the chapter describes additional procedural controls that should exist around a gateway queue manager or an WebSphere MQ Internet pass-thru server. These controls are necessary because a gateway queue manager or WebSphere MQ Internet pass-thru server represents a potential point of attack against your company.

10.1.2 Scenario flow

The scenario uses two applications:

- A service requester application sends a request message to the business partner and later receives a response message.
- A service provider application receives a message from the business partner and processes it.

Service requester message flow

The message flow for Application 1 (service requester) is from the application to an application queue manager, which is as far as the application has visibility. The messages then flow to the gateway queue manager. The gateway queue manager forwards the message via the WebSphere MQ Internet pass-thru proxy to the business partner queue manager. Hopefully, the business partner has implemented a secure connection topology similar to the top part of the picture that is shown in Figure 10-2 on page 145, but this information will not be known by the WebSphere MQ administrator who is setting up this sort of link.

The reply message is returned on a separate channel through the WebSphere MQ Internet pass-thru server to the gateway queue manager.

Service provider message flow

The Application 2 (service provider) message flow is from the business partner to the gateway queue manager through the WebSphere MQ Internet pass-thru proxy. The messages then flow to the application queue manager for delivery to the application. The application processes the message, and sends a reply message back to the gateway queue manager. The gateway queue manager reply path is enforced by a message exit that runs on the gateway queue manager.

Context diagram for B2B messaging

Figure 10-2 on page 145 shows the applications, queue managers, and surrounding infrastructure that are used in the B2B scenario. It shows an ideal environment where all the certificates in use are signed by a certificate authority (CA) and where Certificate Revocation Lists (CRLs) are checked whenever certificates are presented.

As discussed in “Determine certificate DNs for WebSphere MQ Internet pass-thru” on page 163, this scenario does not use CA signed certificates for the gateway queue manager, WebSphere MQ Internet pass-thru service, or the business partner queue manager. Self-signed certificates were used in each of these locations.

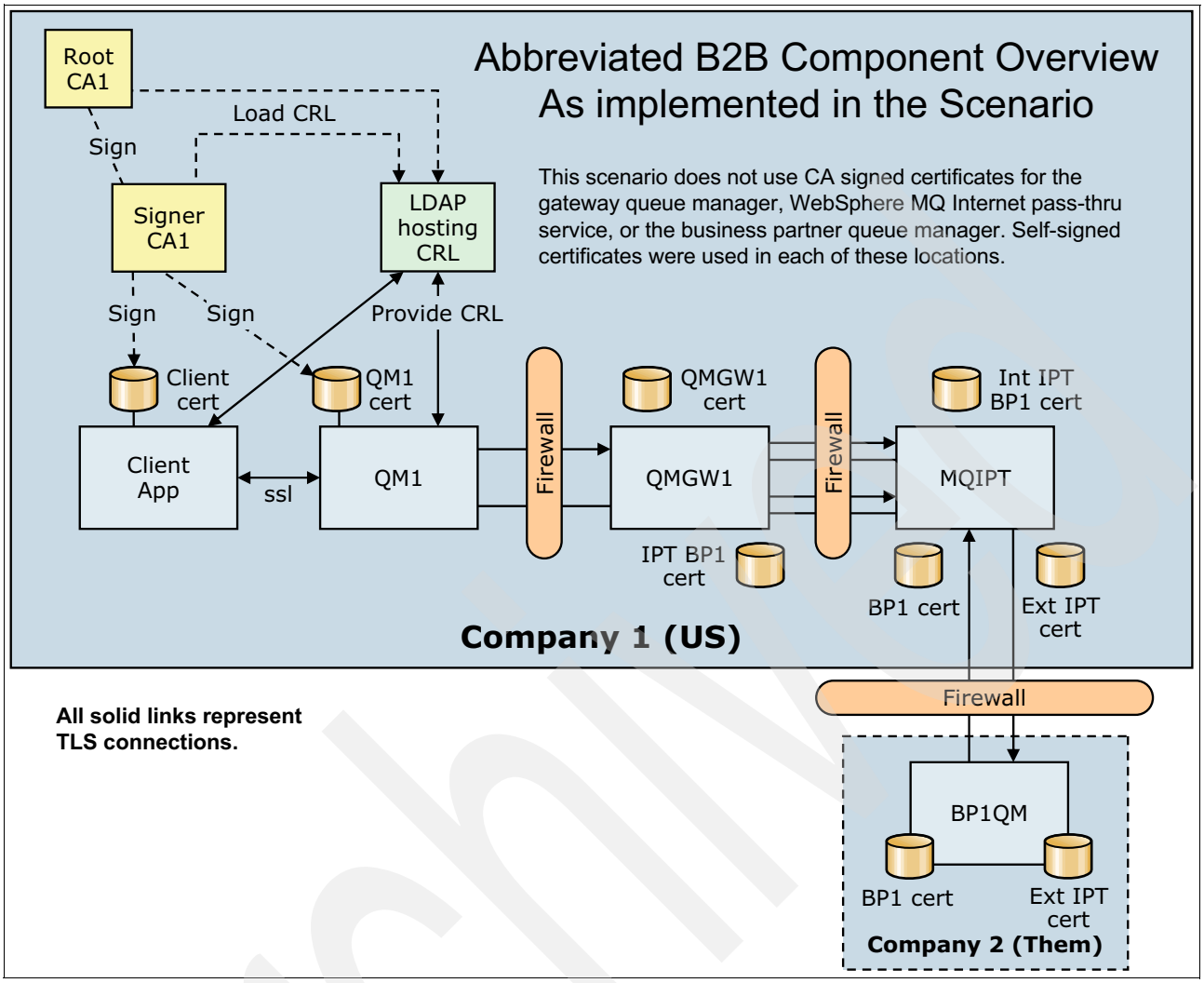


Figure 10-2 Applications, queue managers, and security infrastructure for the B2B scenario

The scenario illustrates the *location hiding*, security controls, and authorization and authentication controls that should be enforced when implementing secure message-based communication between business partners. In this case, there are two partners:

- Company1 (“Us”)
- Company2 (the business partner “Them”)

10.2 WebSphere MQ and WebSphere MQ Internet pass-thru features and practices

Some background on key repositories and CAs is useful to understand the function that is provided by WebSphere MQ Internet pass-thru.

Every reputable CA guarantees that no two certificates will ever use the same DN. Therefore, during the connection handshake, the SSL or TLS server evaluates the DN as a unique identifier. However, there is nothing to prevent two different CAs from issuing certificates bearing the same DN. Therefore, if a key repository contains root signer certificates from

multiple CAs, there is a risk that an unauthorized person could get the second CA to issue a certificate with a matching DN, which would be trusted by the queue manager.

Although this attack is moderately difficult to execute, the possibility of success rises with each CA signer certificate that is added to the keystore. For this reason, a key repository will ideally have only one root signer certificate.

This process presents a difficulty in a B2B situation because each business partner may use a different CA. Because WebSphere MQ uses a single key repository per queue manager, terminating the B2B connections on the queue manager may require a keystore with multiple CA signer root certificates.

Even if the risk of adding multiple signer certificates to the keystore is acceptable, a related issue is that WebSphere MQ supports only one personal certificate, which is the certificate that a queue manager presents to remote partners to identify itself. So when the local and remote queue manager use different CAs, both queue managers are required to trust their own CA as well as the CA of their remote partner.

WebSphere MQ Internet pass-thru addresses these issues by terminating the remote partner's SSL or TLS connection before it reaches the queue manager. Because WebSphere MQ Internet pass-thru allows for separate trust and identity stores on each connection (WebSphere MQ Internet pass-thru calls these connections "*routes*"), a dedicated key repository can be created for each remote business partner where each key repository contains a single CA signer certificate and a personal certificate signed by the same CA as the remote partner. This means that WebSphere MQ Internet pass-thru can authenticate SSL and TLS connections more strongly than can WebSphere MQ in the typical B2B scenario where multiple CAs are used.

10.2.1 Application considerations

One of the objectives of the B2B security design is to isolate the namespaces of the internal messaging network from those of the business partners. For example, it should be possible for the local queue manager and the business partner's queue manager to both have an instance of a queue named `PAYROLL` without causing routing errors. With proper isolation, there is no risk of either party disrupting the other's messaging network. The other business requirement driving this design is to prevent resource enumeration attacks. If it is possible for an attacker to obtain the names of all the queues, topics, or other details of the internal messaging network, this information can be used to escalate and target the attack.

Ideally, the B2B gateway design will use static routing such that every authorized remote and local destination is represented by a QRemote or QAlias. This design creates a dependency on application design. Although it is possible to implement active routing on the gateway by using exits or routing applications, these exits or routing applications represent additional moving parts, more complexity, and a greater attack surface. Addressing this situation in application design is therefore the preferred solution.

Applications that will communicate with external partners should be written with more care and attention than might be given to a normal MQ application program. Most important is that when an application sends a request to a business partner through WebSphere MQ, two aspects of *location hiding* should be implemented:

- The location of the business partner should be hidden by using a QRemote[1] definition to send the message to the gateway. Alternatively, an alias to a cluster queue hosted on the gateway could be used to enable routing through multiple parallel gateway instances.

[1] An MQ object type. It provides a name on the local queue manager that allows a message to be sent to a queue on another queue manager, and provides the route information via the transmission queue name.

Figure 10-3 on page 147 shows the way that the request message flows through the systems from QRemote to QRemote until reaching the destination queue on the business partner queue manager.

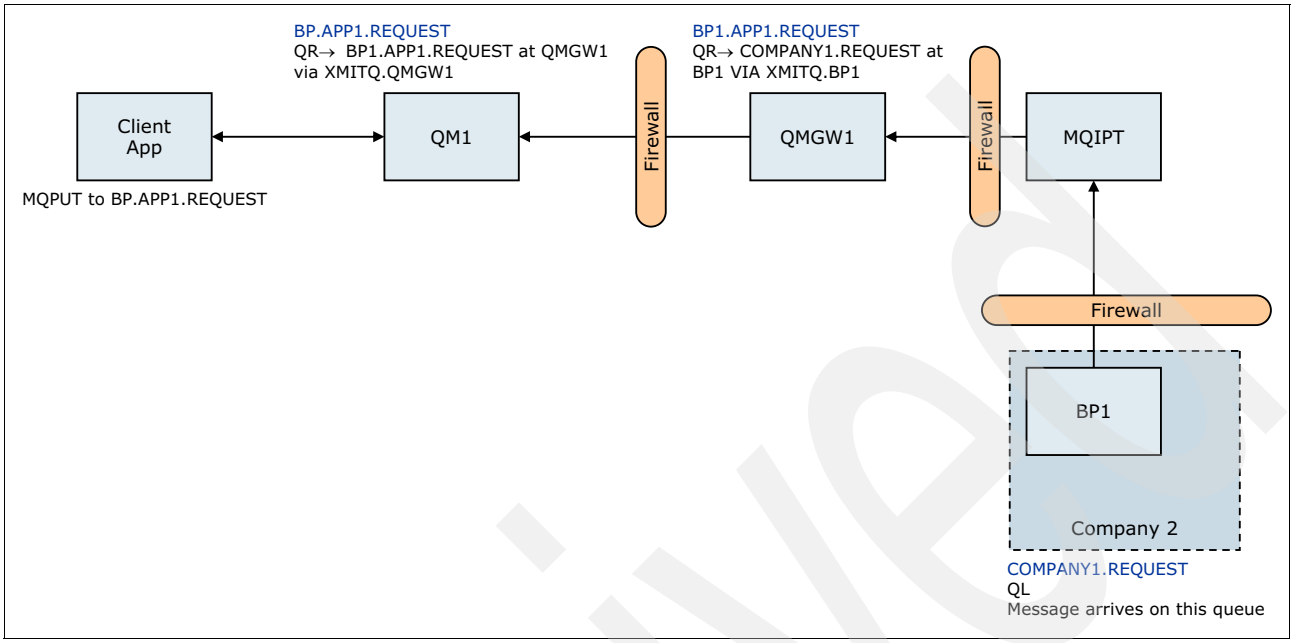


Figure 10-3 Message flow from Application 1 to business partner 1

- The location of the application itself should be hidden from the business partner by using a reply-to queue alias[2]. This technique populates the `ReplyToQMgr` and `ReplyToQ` fields in the MQ Message Descriptor (MQMD). A Reply To Queue Alias (RTQA) is a QRemote definition. The `RNAME` and `RQMNAME` values from the QRemote definition are used by the queue manager to fill in the message descriptor reply values when the request message is sent. For this B2B scenario, the RTQA causes the outbound request message's `ReplyToQMgr` and `ReplyToQ` fields to resolve to a queue on the gateway rather than the actual internal queue and queue manager names. The business partner sees only the gateway queue manager and queue name but inbound messages route through these to the actual internal destination. The application then has to use two different names for its reply queue: one name is the queue that it opens to receive replies, and the second name is the queue name that it puts into the MQMD of the request (the RTQA). Figure 10-4 on page 148 shows the relationships between the reply and queue manager aliases, and the QRemote definitions that route messages to the correct destination.

  In the absence of the application implementing this *location hiding*, a message exit implemented at the gateway can perform this function. A sample exit to perform this function and some other important functions is used in this scenario. The exit is included in Appendix A, "Working with the itsoME message exit" on page 263.

[2] A QRemote definition used in the message descriptor of a request message. The rname (remote name) and rqmname (remote queue manager name) values from the QRemote definition are substituted into the message descriptor in place of the original replytoq and replytoqm values.

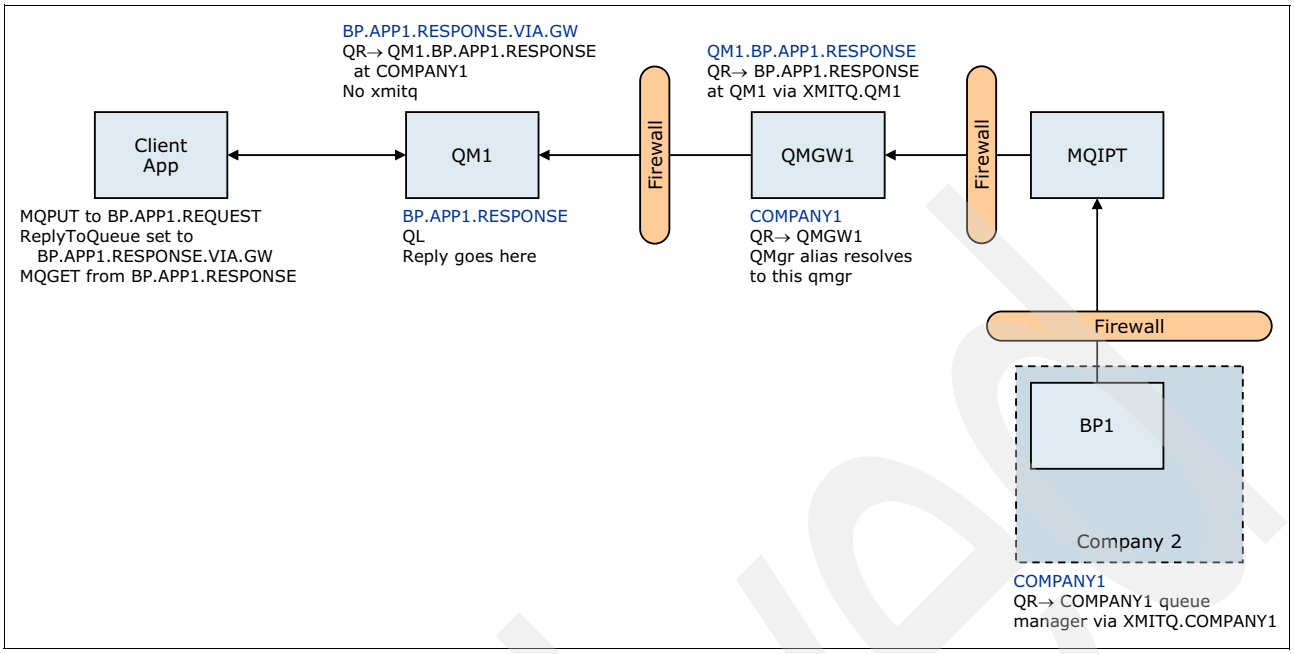


Figure 10-4 Reply to queue alias and reply queue path from Business Partner 1 to Application 1

Applications that receive messages from business partners and process them have different requirements. *Location hiding* is implemented by the gateway alone, but the application needs to implement more defensive checking of message content than would be the case for receiving a message from another application within the same business. Illustrating the additional checking is outside the scope of this book. It could be written into the application itself, or it could be implemented in a dedicated B2B application located between the gateway queue manager and the end business application. This B2B application could also perform external/internal format conversion as well as the heightened format/value validation of the incoming message.

10.2.2 Application queue manager

In this scenario, an application queue manager provides messaging services to the business application at our company (Company1). There are two applications that use this queue manager:

- Company1 BP service requester application. This application sends messages to a business partner (Company2) and expects to receive a reply. It puts the request message to a QRemote, using an RTQA in the ReplyToQ of the MQMD. The application opens a QLocal[3] to receive replies. The infrastructure to link the RTQA and the QLocal is in both the application queue manager and the gateway queue manager. See Figure 10-4.
- Company1 BP service provider application. This application receives request or datagram (notification) messages from the business partner, and processes them. It will send a reply if one is requested and appropriate for the message type.

[3] An MQ object type. It is a place on the queue manager where messages can be stored.

10.2.3 Gateway queue manager

The gateway queue manager is a critical component of the secure connection between Company1 and its business partner Company2. It is more carefully secured and managed than a normal application queue manager, with additional attention paid to change and security controls. It has no application local queues, and no applications connecting to it.

The gateway queue manager provides three major functions:

- QRemote definition to pass a message from a local application on to the business partner queue manager
- QRemote definition to pass a message from the business partner on to the application queue manager
- Channel message exits to ensure that unwanted WebSphere MQ capabilities cannot reach our application queue manager. These exits also ensure that the internal queue manager information related to Company1 is hidden from Company2.

In addition to the normal secure queue manager configuration (see Chapter 6, “WebSphere MQ security controls” on page 41), a gateway queue manager should be further protected by the following measures:

- No low-risk changes permitted. All change to the gateway queue manager must show elevated risk and be processed through full peer and change approval board reviews.
- Updates to the gateway queue manager are only permitted during approved change windows.
- The gateway queue manager runs on a dedicated server.
- There is no general access to the gateway queue manager server. Only system administrators and MQ administrators have any access to the server.
- All connectivity to the gateway queue manager is authenticated using SSL/TLS certificates, with strong SSLPEER checking.
- No MQ client access to the gateway queue manager is permitted.
- After any update during setup or for a change, all configuration files and keystores are set to read-only (even by the `mqm` user).

10.2.4 WebSphere MQ Internet pass-thru server

The WebSphere MQ Internet pass-thru server is a proxy server for the WebSphere MQ protocol. In this scenario, it is placed in the outermost DMZ between Company1 and its business partner Company2. This is a normal location of a WebSphere MQ Internet pass-thru server. Its function is to terminate SSL/TLS connections between the companies in the DMZ, and to manage the ongoing SSL/TLS connections into the gateway queue manager, or to the business partner.

The connection via WebSphere MQ Internet pass-thru is a normal MQ protocol connection. We do not use the http tunneling feature of WebSphere MQ Internet pass-thru in this scenario. WebSphere MQ Internet pass-thru is used in order to gain better control of the trust for validating partner SSL/TLS certificates.

Like a gateway queue manager server, the WebSphere MQ Internet pass-thru server is a potential attack point for your company. As such, the additional controls that apply to the gateway queue manager server should also apply to WebSphere MQ Internet pass-thru servers. See the introductory material in 10.2.3, “Gateway queue manager” on page 149.

The WebSphere MQ Internet pass-thru server acts as a protocol proxy by mapping an incoming port to an outgoing address and port. It performs full WebSphere MQ channel protocol validation and SSL/TLS termination in both directions.

The scenario presents a tested configuration that could easily be extended to support multiple external business partners, or multiple gateway queue managers communicating through the WebSphere MQ Internet pass-thru server. Enabling high availability or high throughput in the gateway layer is a more extensive challenge, and is beyond the scope of this book.

WebSphere MQ Internet pass-thru is a stateless proxy between sessions. When establishing a new session (channel) between two queue managers, a network load balancer could be used to select between any number of available WebSphere MQ Internet pass-thru servers in the DMZ. Topology can be used to provide high availability to this layer. The specific WebSphere MQ Internet pass-thru server selected during setup of a specific channel is transparent to the queue managers. If the WebSphere MQ Internet pass-thru server were to fail, the channel would stop. WebSphere MQ channel retry processing will restart the channel. The network load balancer can direct the connection request to an active WebSphere MQ Internet pass-thru server.

This capability is important because it means that several WebSphere MQ Internet pass-thru servers can be used to provide a highly available proxy solution. The scenario that is presented does not include this type of highly available configuration.

10.3 Implementing the scenario

There are many distinct objects that need to be created or configured in this scenario in order to enable communication between the business partners. Each of these objects is discussed and illustrated in its own section. There is also a section covering environmental controls, such as firewalls, which will need to be considered.

One of the major differences between a B2B configuration and an internal request/response scenario is the way that reply-to queues are handled and secured. The normal convention for sending responses is to permit the server application to put response messages using the queue manager alias[4] for the requesting queue manager.

In this B2B scenario, this alias is not allowed. All messages are put to queues that resolve on the local queue manager. An exit is used to replace the reply-to queue information if the application does not use Reply to queue aliases. Instead of granting the server application the right to put messages to any queue on the gateway queue manager, the specific right to put to the local queue definition is granted.

10.3.1 Application queue manager configuration

The application queue manager provides messaging services to the application program. In this scenario, the application is taken from a SupportPac. Clearly in an actual client configuration, it would be an application that might be running under WebSphere Application Server or another application runtime environment.

[4] A QRemote definition without a name for the queue at the other queue manager. This alias is used to resolve queue manager names to the transmission queue used to reach them, and for other name resolution functions.

Follow these steps for configuring the application queue manager:

1. Create and configure the application queue manager using basic and administration hardening (Chapter 6, “WebSphere MQ security controls” on page 41). In this scenario, the queue manager is called `QM1`. It runs on server `wmqsvr1` and will listen on port 1414.
2. Configure the queue manager to validate certificates, including checking certificate revocation using Certificate Revocation Lists via Lightweight Directory Access Protocol (LDAP) (CRLs). Any Online Certificate Status Protocol (OCSP) checking will be driven by Authority Information Access (AIA) information within the certificates. LDAP-based (rather than OCSP) checking is illustrated because OCSP is not available on all operating systems that support WebSphere MQ. See Chapter 12, “Scenario: CRL/OCSP certificate revocation” on page 219 for further information about CRL and OCSP.
3. Configure the user IDs and groups for the applications. Example 10-1 shows the creation of the user IDs that are specified in the MCAUSER parameter of svrconn channels. The groups are used to hold the authorizations granted to the users. The users are set up with a dummy shell to prevent them from being used as a way to get access to the host. The `root` user has to create users and groups.

   Example 10-1 Users and groups for client Message Channel Agent (MCA) users

   ```
   groupadd mcaappg1
   useradd -g mcaappg1 -s /bin/false mcaapp1
   groupadd mcaappg2
   useradd -g mcaappg2 -s /bin/false mcaapp2
   ```

4. Configure the user ID and group for the gateway queue manager receiver channel. Example 10-2 shows the creation of a user and group to be used for controlling security of the incoming channel from the gateway queue manager. Again, the user ID is blocked using an invalid shell.

   Example 10-2 Users and groups for receiver channel from the gateway queue manager

   ```
   groupadd mcaqmgw1
   useradd -g mcaqmgw1 -s /bin/false mcaqmgw1
   ```

5. Specify the application certificate DN. Example 10-3 shows the DNs that the queue manager will expect to see when validating application certificates. Creation of the keystores and configuration of the client environment are discussed in 10.3.4, “Application configuration” on page 177.

   Example 10-3 Client certificate DNs

   ```
   CN=Requester Application 1,OU=SA-W216 Residency,O=IBM ITSO,C=US
   CN=Server Application 1,OU=SA-W216 Residency,O=IBM ITSO,C=US
   ```

6. Add the gateway queue manager self-signed certificate (created in Example 10-15 on page 156) to the queue manager Certificate Management Services (CMS) keystore. The commands to perform this task are shown in Example 10-4.

   Example 10-4 Add trust for the gateway queue manager to the CMS keystore

   ```
   cd /var/mqm/qmgrs/QM1/ssl

   runmqakm -cert -add -db key.kdb -stashed -file QMGW1.cer -trust enable -label
   "QMGW1 self signed"
   ```

7. Create the application client channels. Example 10-5 shows both svrconn channels for the queue manager, and the clntconn channels for building a client channel connection table (CCDT).

Example 10-5 Commands to enable client connections to the application queue manager

```
runmqsc QM1
   define channel(B2BAPP1.SVRCONN) chltype(svrconn) trptype(tcp) +
   descr('Application B2BApp1 connect here') +
   mcauser('mcaapp1') +
   maxmsgl(4194304) +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=Requester Application 1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   replace

   define channel(B2BAPP2.SVRCONN) chltype(svrconn) trptype(tcp) +
   descr('Application B2BApp2 connect here') +
   mcauser('mcaapp2') +
   maxmsgl(4194304) +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=Server Application 2,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   replace

   define channel(B2BAPP1.SVRCONN) chltype(clntconn) trptype(tcp) +
   descr('Application B2BApp1 gets QM1 info from here') +
   conname('wmqsvr1.saw216.itso.ibm.com(1414)') +
   qmname('App1QM1') +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=QM1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   maxmsgl(4194304) +
   replace

   define channel(B2BAPP2.SVRCONN) chltype(clntconn) trptype(tcp) +
   descr('Application B2BApp2 gets QM1 info from here') +
   conname('wmqsvr1.saw216.itso.ibm.com(1414)') +
   qmname('App2QM1') +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=QM1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   maxmsgl(4194304) +
   replace

   end
```

8. Create the channel infrastructure for connection to the gateway queue manager. Example 10-6 shows the **runmqsc** commands to create the channels and associated queues for communication between the core application queue manager and the gateway queue manager.

Example 10-6 Commands to enable message flows between core and gateway queue managers

```
runmqsc QM1
   define qlocal(XMITQ.QMGW1) usage(xmitq) +
   descr('Messages for the B2B Gateway QM') +
   initq(SYSTEM.CHANNEL.INITQ) trigger +
   trigtype(first) trigdata(QM1.QMGW1) +
   replace
```

```
   define channel(QM1.QMGW1) chltype(sdr) trptype(tcp) +
   xmitq(XMITQ.QMGW1) +
   conname('wmqsvr4.saw216.itso.ibm.com(1415)') +
   locladdr('wmqsvr1.saw216.itso.ibm.com') +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=QMGW1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   monchl(low) +
   replace

   define channel(QMGW1.QM1) chltype(rcvr) trptype(tcp) +
   mcauser('mcaqmgw1') putaut(def) +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=QMGW1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   sslcauth(required) +
   monchl(low) mrtmr(1000) mrrty(3600) +
   replace

   end
```

9. Create the QRemote definition for sending messages to the gateway. Example 10-7 shows the **runmqsc** commands to create the qremote that the application will use to send messages.

Example 10-7 Application queue to send messages to the business partner

```
runmqsc QM1
   define QRemote(BP.APP1.REQUEST) +
   descr('Send application requests to Business Partner via GW') +
   rqmname(QMGW1) rname(BP1.APP1.REQUEST) +
   xmitq(XMITQ.QMGW1) +
   replace

   end
```

10. Create the QLocal definition for receiving response messages. Example 10-8 shows the **runmqsc** command to create the response queue that is used by the application.

Example 10-8 Command to create the response queue

```
runmqsc QM1
   define QLocal(BP.APP1.RESPONSE) +
   usage(normal) +
   descr('Response messages for App1 from our business partner') +
   monq(qmgr) +
   replace

   end
```

11. Create a QRemote definition to act as the ReplyToQAlias for the response queue. Example 10-9 shows the command to create a queue definition that will act together with a definition on the gateway queue manager to return messages back to the application. Using this alias queue as the reply-to queue means that a broader security rule allowing access to a queue manager alias can be avoided on the gateway queue manager. The reply to queue manager name is not `QMGW1`, but a very generic alias (`COMPANY1`). This alias will be resolved to `QMGW1` on the way back from the business partner.

Example 10-9 Command to create a replytoq alias

```
runmqsc QM1
   define QRemote(BP.APP1.RESPONSE.VIA.GW) +
   descr('Direct the Business Partner to respond via GW') +
   rqmname(COMPANY1) rname(QM1.BP.APP1.RESPONSE) +
   xmitq(' ') +
   replace

   end
```

12. Create the QLocal definition to receive request, action, and notification messages from the business partner. Example 10-10 shows the creation of a local queue that Application 2 uses to receive request messages.

Example 10-10 Command to create local queue for request messages to Application 2

```
runmqsc QM1
   define QLocal(BP.APP2.REQUEST) +
   usage(normal) +
   descr('Request messages for App2 from our business partner') +
   monq(qmgr) +
   replace

   end
```

13. Create a QRemote definition to send replies for the business partner back to the gateway queue manager as shown in Example 10-11. A remote queue definition is used in order to allow stronger security controls than would be possible if using a queue manager alias to send the response back. A queue manager alias (LOCALQM) is defined to force the name resolution to be localized to the application queue manager. An exit is used on the gateway queue manager to replace the ReplyTo information from the business partner with the names that are created in this example.

Example 10-11 Create QRemotes to send B2B replies back to the gateway queue manager

```
runmqsc QM1
   define QRemote(BP1.FOREIGN.RESPONSE.QUEUE) +
   descr(‘Name of business partners RQ, prefixed with BP QM name’) +
   rqmname(QMGW1) rname(BP1.FOREIGN.RESPONSE.QUEUE) +
   xmitq(XMITQ.QMGW1) +
   replace

   define QRemote(LOCALQM) +
   descr('QM Alias to force local name resolution for BP messages') +
   rqmname(QM1) rname(' ') +
   xmitq(' ') +
   replace

   end
```

14. Grant authorizations to application groups as shown in Example 10-12. These authorizations could also be implemented using **SET AUTHREC** commands in runmqsc instead.

Example 10-12 Authorize client applications to use WebSphere MQ objects

```
setmqaut -m QM1 -t qmgr -g mcaappg1 +connect +inq
setmqaut -m QM1 -t qmgr -g mcaappg2 +connect +inq
setmqaut -m QM1 -t q -n BP.APP1.REQUEST -g mcaappg1 +put +inq
setmqaut -m QM1 -t q -n BP.APP1.RESPONSE -g mcaappg1 +get +browse +inq
setmqaut -m QM1 -t q -n BP.APP2.REQUEST -g mcaappg2 +get +browse +inq
setmqaut -m QM1 -t q -n BP1.FOREIGN.RESPONSE.QUEUE -g mcaappg2 +put +inq
```

15. Grant authorizations to the gateway channel group as shown in Example 10-13. These authorizations allow the channel receiving messages from the gateway queue manager the minimum capability to perform to requirements.

Example 10-13 Authorize channel from gateway to access QM1 objects

```
$ setmqaut -m QM1 -t qmgr -g mcaqmgw1 +connect +inq +setall -set
$ setmqaut -m QM1 -t q -n DLQ -g mcaqmgw1 +put +inq +setall
$ setmqaut -m QM1 -t q -n BP.APP1.RESPONSE -g mcaqmgw1 +put +inq +setall
$ setmqaut -m QM1 -t q -n BP.APP2.REQUEST -g mcaqmgw1 +put +inq +setall
```

10.3.2 Gateway queue manager configuration

The gateway queue manager acts as a bridge between the application queue manager and the external world. It does not perform any application processing. Its role is to receive messages from one side, and pass them on to the other. During this process, the gateway queue manager protects the application queue manager from flooding by the partner. It provides transparency of location, and can be used to host channel message exits that can enforce information hiding about the application queue managers.

The configuration of a gateway queue manager requires some elements that are different from a typical queue manager configuration, such as the following elements:

- No dead-letter queue (DLQ). We want to block a partner that is sending messages without a valid destination. If we do not block such a partner, it could be used to implement a more widespread denial-of-service attack (DoS).
- Tight security lockdown of not only the queue manager, but also the server.
- Tight change control. Low-impact changes are not permitted. All changes must be fully reviewed and approved at the Change Approval Board level.
- Small maximum depths on queues, which helps to prevent flooding by badly behaved partner queue managers, and ensures that in the event of an application queue manager outage, we are not accepting messages that cannot be handled.
- Separation of listeners for internal, external, and administration traffic.
- Message exits on incoming and outgoing channels to allow enforcement of security standards.
- No application access (client or server bindings). Only administrative access is allowed.
- Administrative access is restricted.
- Explicitly set security-related options, such as PUTAUT.

- Certificate providers and cipher suites are chosen that work with WebSphere MQ Internet pass-thru. (In the scenario, we used self-signed certificates because the SHA2-signed CA certificates from our CA were not recognized by the WebSphere MQ Internet pass-thru crypto library.)
- sslfips(no) setting. None of the FIPS-compliant algorithms work with the current version of WebSphere MQ Internet pass-thru (version 2.0.0.3).

These steps are used to configure the gateway queue manager:

1. Create the gateway queue manager, in this scenario, `QMGW1`. It listens for connections from the internal network interface on port 1415 and for connections from WebSphere MQ Internet pass-thru on the external interface using port 1416. The queue manager runs on server `wmqsvr4`.
2. Set up certificate stores and trusted CA certificates as illustrated in 8.11, “WebSphere MQ (CMS) keystores” on page 92. The DN of the certificate for `QMGW1` is shown in Example 10-14.

   Example 10-14 DN for the certificate for QMGW1

   ```
   CN=QMGW1,OU=SA-W216 Residency,O=IBM ITSO,C=US
   ```

3. Alter the queue manager to remove the FIPS requirement by using the following **runmqsc** command:

   ```
   alter qmgr sslfips(no)
   ```

4. Alter the queue manager to remove DLQ processing by using the following **runmqsc** command:

   ```
   alter qmgr deadq(’ ’)"
   ```

5. Create a self-signed certificate as illustrated in Example 10-15.

   Example 10-15 Create self-signed certificate for gateway queue manager

   ```
   cd /var/mqm/qmgrs/QMGW1/ssl

   runmqakm -fips -cert -create -db key.kdb -stashed -label ibmwebspheremqqmgw1
   -dn "CN=QMGW1,OU=SA-W216 Residency,O=IBM ITSO,C=US" -sigalg SHA1WithRSA -size
   2048

   runmqakm -fips -cert -extract -db key.kdb -stashed -label ibmwebspheremqqmgw1
   -file QMGW1.cer
   ```

6. Send the extracted certificate file (`qm1.cer`) to the WebSphere MQ administrator for the QM1 (application hosting) queue manager. The administrator will need it in order to complete Example 10-4 on page 151.
7. Configure the queue manager to validate certificates, including checking certificate revocation by using CRLs. The CRLs will be retrieved via LDAP from one or more LDAP servers defined using the queue manager SSLCRLNL property. Any OCSP checking will be driven by AIA information within the certificates.

   LDAP rather than OCSP checking is currently preferred because OCSP is not available on all systems that support WebSphere MQ. If the certificate AIA section is present and contains OCSP information, OCSP will be used in preference to LDAP if the servers can be reached. The configuration that is needed is described in 12.4.1, “Configuring OCSP in WebSphere MQ” on page 228.

8. Create the user ID and group for the application queue manager receiver channel as shown in Example 10-16. The user is locked down with an invalid shell to prevent unintended access.

Example 10-16 Create user and group for channel from qm1 to gateway queue manager

```
groupadd mcaqm1
useradd -g mcaqm1 -s /bin/false mcaqm1
```

9. Create the user ID and group for the business partner queue manager receiver channels as shown in Example 10-17.

Example 10-17 Create users and groups for channels from business partners

```
groupadd mcacustg1
useradd -g mcacustg1 -s /bin/false mcacust1
groupadd mcacustg2
useradd -g mcacustg2 -s /bin/false mcacust2
```

10. Specify the business partner certificate DN. Example 10-18 shows the names that are used in the scenario. These certificates are self-signed and are implemented in the WebSphere MQ Internet pass-thru server. Self-signed certificates were used, not because of a preference, but because the CA that we used had a signing certificate with the SHA2 signature algorithm. We found that WebSphere MQ Internet pass-thru was not able to use keystores that contained certificates in this format.

Example 10-18 DNs used by WebSphere MQ Internet pass-thru to assert business partner identity

```
CN=Business Partner 1,OU=SA-W216 Residency,O=IBM ITSO,C=US
CN=Business Partner 2,OU=SA-W216 Residency,O=IBM ITSO,C=US
```

11. Configure the listeners for administration, internal connections, and external (WebSphere MQ Internet pass-thru) connections as shown in Example 10-19. Separate listeners are provided for each connection. The listeners use different ports and are bound to the specific interface for their use.

Example 10-19 Listeners that are needed in the gateway queue manager

```
runmqsc QMGW1
   define listener(LISTENER.ADMIN) trptype(tcp) +
   descr('Listener on admin interface for Administration only') +
   ipaddr('wmqsvr4-m.saw216.itso.ibm.com') +
   port(14141) +
   control(qmgr) +
   replace

   define listener(LISTENER.INTERNAL) trptype(tcp) +
   descr('Listener on internal interface for Core access only') +
   ipaddr('wmqsvr4.saw216.itso.ibm.com') +
   port(1415) +
   control(qmgr) +
   replace

   define listener(LISTENER.EXTERNAL) trptype(tcp) +
   descr('Listener on external interface for MQIPT traffic only') +
   ipaddr('wmqsvr4-e.saw216.itso.ibm.com') +
   port(1416) +
   control(qmgr) +
```

```
   replace

   end
```

12. Configure the channels to interact with the application queue manager as shown in Example 10-20 on page 158.

    Note the very small maxdepth (2.5 times the channel batch size). This size is because there is no application processing on the gateway. Messages coming in will be forwarded immediately if the target is available. If the target queue manager is not available, we do not want to hide the outage by queuing thousands or tens of thousands of messages in the gateway queue manager. The small maxdepth combined with the absence of a DLQ and message retry values ensures that messages stay in sequence, are not lost into a DLQ, and that upstream outages become visible.

Example 10-20 Set up channels for application queue manager communication

```
runmqsc QMGW1
   define qlocal(XMITQ.QM1) usage(xmitq) +
   descr('Messages for the B2B Application 1 QM') +
   initq(SYSTEM.CHANNEL.INITQ) trigger +
   trigtype(first) trigdata(QMGW1.QM1) +
   maxdepth(25) +
   replace

   define channel(QMGW1.QM1) chltype(sdr) trptype(tcp) +
   xmitq(XMITQ.QM1) +
   conname('wmqsvr1.saw216.itso.ibm.com(1414)') +
   locladdr('wmqsvr4.saw216.itso.ibm.com') +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=QM1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   monchl(low) +
   replace

   define channel(QM1.QMGW1) chltype(rcvr) trptype(tcp) +
   mcauser('mcaqm1') putaut(def) +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TLS_RSA_WITH_AES_256_CBC_SHA) +
   sslpeer('CN=QM1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   sslcauth(required) +
   monchl(low) mrtmr(1000) mrrty(3600) +
   msgexit('/var/mqm/exits64/itsoME(MsgExit)') +
   msgdata('MCA/ALL/RTQM=COMPANY1/') +
   replace

   end
```

13. Configure the channels to interact with the Business Partner 1 queue manager (`BP1`) through WebSphere MQ Internet pass-thru as shown in Example 10-21 on page 159. These channels direct traffic to the WebSphere MQ Internet pass-thru server, but the channel name reflects the partner queue manager, not the WebSphere MQ Internet pass-thru server.

    In this scenario, triple DES encryption is used because WebSphere MQ 7.5 TLS validation does not accept TLS cipher suites when communicating with WebSphere MQ Internet pass-thru.

For a description, see the WebSphere MQ 7.5 Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy12870_1.htm

Because the current release (v2.0.0.3) of WebSphere MQ Internet pass-thru only uses SSL v3, only those CipherSpecs that are listed as using SSL 3.0 in the table can be used between WebSphere MQ v7.5 and WebSphere MQ Internet pass-thru. For more information about an APAR to address this issue, see 10.5, “Advanced Encryption Standard support in WebSphere MQ Internet pass-thru” on page 183. After the issue of TLS support has been addressed, cipher specs associated with the TLS 1.0 protocol will be available if they are listed in the MQIPT manual as a supported algorithm.

The configuration shows the use of an exit to massage incoming messages. This exit provides protection against Report Message-based attacks (see 6.6.5, “Careful use of report messages” on page 70) and forces reply-to queues to be resolved locally, rather than using queue manager aliases. The exit is included and discussed in Appendix A, “Working with the itsoME message exit” on page 263.

Example 10-21 Create channels for communication with a business partner via MQIPT

```
runmqsc QMGW1
   define qlocal(XMITQ.BP1) usage(xmitq) +
   descr('Messages for the Business Partner 1 QM') +
   initq(SYSTEM.CHANNEL.INITQ) trigger +
   trigtype(first) trigdata(QMGW1.BP1) +
   maxdepth(25) +
   replace

   define channel(QMGW1.BP1) chltype(sdr) trptype(tcp) +
   xmitq(XMITQ.BP1) +
   conname('wmqiptsvr1-e.saw216.itso.ibm.com(5001)') +
   locladdr('wmqsvr4-e.saw216.itso.ibm.com') +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TRIPLE_DES_SHA_US) +
   sslpeer('CN=Business Partner 1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   monchl(low) +
   replace

   define channel(BP1.QMGW1) chltype(rcvr) trptype(tcp) +
   mcauser('mcacust1') putaut(def) +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TRIPLE_DES_SHA_US) +
   sslpeer('CN=Business Partner 1,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   sslcauth(required) +
   monchl(low) mrtmr(1000) mrrty(3600) +
   msgexit('/var/mqm/exits64/itsoME(MsgExit)') +
   msgdata('MCA/ALL/RTQM=LOCALQM/') +
   replace

   end
```

The channel configuration can be repeated (with appropriate changes where needed) for each business partner connection. Each business partner will use a different port on the WebSphere MQ Internet pass-thru server. Configuration of the channels to interact with the Business Partner 2 queue manager through WebSphere MQ Internet pass-thru is shown in Example 10-22. This example is effectively a duplicate of Example 10-21, showing the changes that are needed for connection to a different business partner. (The changes are highlighted in bold).

Example 10-22 Create channels for a second business partner

```
runmqsc QMGW1
   define qlocal(XMITQ.BP2) usage(xmitq) +
   descr('Messages for the Business Partner 2 QM') +
   initq(SYSTEM.CHANNEL.INITQ) trigger +
   trigtype(first) trigdata(QMGW1.BP2) +
   maxdepth(25) +
   replace

   define channel(QMGW1.BP2) chltype(sdr) trptype(tcp) +
   xmitq(XMITQ.BP2) +
   conname('wmqiptsvr1-e.saw216.itso.ibm.com(5002)') +
   locladdr('wmqsvr4-e.saw216.itso.ibm.com') +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TRIPLE_DES_SHA_US) +
   sslpeer('CN=Business Partner 2,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   monchl(low) +
   replace

   define channel(BP2.QMGW1) chltype(rcvr) trptype(tcp) +
   mcauser('mcacust2') putaut(def) +
   batchsz(10) npmspeed(normal) usedlq(no) +
   sslciph(TRIPLE_DES_SHA_US) +
   sslpeer('CN=Business Partner 2,OU=SA-W216 Residency,O=IBM ITSO,C=US') +
   sslcauth(required) +
   monchl(low) mrtmr(1000) mrrty(3600) +
   msgexit('/var/mqm/exits64/itsoME(MsgExit)') +
   msgdata('MCA/ALL/RTQM=LOCALQM/') +
   replace

   end
```

14. Configure the queues to forward messages to the business partner, and to forward messages from the business partner to the application queue manager. Example 10-23 shows the configuration for Application 1 (send requests and receive responses).

Example 10-23 Queues for forwarding messages required by Application 1

```
runmqsc QMGW1
   define QRemote(BP1.APP1.REQUEST) +
   descr('Forward from APP1 to BP1 Request queue') +
   rqmname(BP1) rname(COMPANY1.REQUESTS) +
   xmitq(XMITQ.BP1) +
   replace

   define QRemote(QM1.BP.APP1.RESPONSE) +
   descr('Forward from BP1 to APP1 Response queue’) +
   rqmname(QM1) rname(BP.APP1.RESPONSE) +
   xmitq(XMITQ.QM1) +
   replace

   end
```

Example 10-24 shows the configuration for Application 2 (receive requests and send responses).

Example 10-24 Queues for forwarding messages required by Application 2

```
runmqsc QMGW1
   define QRemote(BP1.APP2.REQUEST) +
   descr('Forward from BP1 to APP2 Request queue') +
   rqmname(QM1) rname(BP.APP2.REQUEST) +
   xmitq(XMITQ.QM1) +
   replace

   define QRemote(BP1.FOREIGN.RESPONSE.QUEUE) +
   descr('Forward from APP2 to BP1 Response queue') +
   rqmname(BP1) rname(FOREIGN.RESPONSE.QUEUE) +
   xmitq(XMITQ.BP1) +
   replace

   end
```

15. Configure a queue manager alias so that COMPANY1, which is what our business partner will see as our reply queue manager name, maps to the gateway queue manager. The commands are shown in Example 10-25. In an environment where messages from the gateway queue manager flow to application queue managers over cluster queues, this alias would map to spaces (' ' characters) instead. Cluster resolution of the resolved queue name is only possible if the effective queue manager name is blank.

Example 10-25 Create Queue manager alias to deal with responses from a business partner

```
define qr(COMPANY1) +
descr('QMgr alias to allow resolution of msgs from partners') +
rname(' ') rqmname(QMGW1) xmitq(' ') +
replace
```

16. Grant authorizations to allow the channel MCAUSER IDs to connect to the queue manager and operate on queues. Example 10-26 shows the commands to grant authority to the MCAUSERID for channel `QM1.QMGW1`. This channel receives messages from the application queue manager. The user ID is `mcaqm1`, which is a member of group `mcaqm1`. Authorities are granted to the group, not to the user, for WebSphere MQ on UNIX style platforms.

Example 10-26 Grant access to allow channel from QM1 to operate

```
setmqaut -m QMGW1 -t qmgr -g mcaqm1 +connect +inq +setall
setmqaut -m QMGW1 -t q -n BP1.APP1.REQUEST -g mcaqm1 +inq +put +setall
setmqaut -m QMGW1 -t q -n BP1.FOREIGN.RESPONSE.QUEUE -g mcaqm1 +inq +put
+setall
```

Example 10-27 shows the access that is required by the `BP1.QMGW1` channel, which receives messages from Business Partner 1 that are destined for applications that are run by Company1.

Example 10-27 Grant access to allow channel from business partner to operate

```
setmqaut -m QMGW1 -t qmgr -g mcacustg1 +connect +inq +setall
setmqaut -m QMGW1 -t q -n QM1.BP.APP1.RESPONSE -g mcacustg1 +inq +put +setall
setmqaut -m QMGW1 -t q -n BP1.APP2.REQUEST -g mcacustg1 +inq +put +setall
```

10.3.3 WebSphere MQ Internet pass-thru configuration

WebSphere MQ Internet pass-thru is a SupportPac that can be downloaded from the WebSphere MQ Support website:

http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24006386&loc=en_US&cs=utf-8&lang=en

Be sure to download the correct version for your target platform and install it according to the instructions in the manual. The manual can be downloaded as a PDF from the same site. As of July 2012, the current version of the ms81 SupportPac is 2.0.0.3.

On Linux, installation is performed by the `root` user ID. Subsequent configuration and run time in the example are shown using the `mqm` user ID and `mqm` group, but this user ID could be another (non-`root`) service account instead. Access to the run time account needs to be limited to authorized users only.

Also, download and install the latest (v7.5) WebSphere MQ Client software for your platform. WebSphere MQ Client is available as a SupportPac:

http://www-304.ibm.com/support/docview.wss?uid=swg24032744

Install the full WebSphere MQ Client, including Global Secure ToolKit (GSKit) and Java Runtime Environment (JRE). The GSKit8, which is accessed with the **`runmqakm`** command-line interface, provides FIPS-compliant key tools. The WebSphere MQ JRE provides the IBM Java Runtime Environment with Java Secure Socket Extension (JSSE) providers supporting the MQ encryption algorithms that are already configured (IBM JSSE Provider 2). It may be possible to use other JREs with the appropriate configuration of JSSE providers.

You should script the start-up of WebSphere MQ Internet pass-thru and have it start automatically at the operating system start-up. The mechanism to implement this function varies by operating system and by whether you are running cluster management software or other automation software on your system. The scripts that are used in this scenario are shown in Appendix B, “Additional tooling for WebSphere MQ Internet pass-thru” on page 291.

Configuration of WebSphere MQ Internet pass-thru requires the configuration file and associated certificate private keys to be held on the file system. Your server configuration must protect these files as strongly as possible. The configuration directory should have permissions set to 700 to minimize exposure of these critical files.

After configuration, all key and configuration files and directories are set to read-only. Allow updates to these files only during subsequent approved change windows.

For this scenario, WebSphere MQ Internet pass-thru is installed on the server `wmqiptsvr1`. Follow these steps:

1. Install the WebSphere MQ client

   The WebSphere MQ Client installation for Linux, as illustrated in Example 10-28, is simple. The installation consists of copying the installer to the machine, unpacking it, and using the **`rpm`** command to install the required components. The specific version illustrated in Example 10-28 is the 64-bit version of WebSphere MQ Client 7.5. The commands are run as `root`.

Example 10-28 Install WebSphere MQ Client components

```
tar zxf CI79SML.tar.gz
./mqlicense.sh -accept
rpm -ivh MQSeriesRuntime-7.5.0-0.x86_64.rpm MQSeriesJRE-7.5.0-0.x86_64.rpm
MQSeriesGSKit-7.5.0-0.x86_64.rpm MQSeriesClient-7.5.0-0.x86_64.rpm
MQSeriesMan-7.5.0-0.x86_64.rpm
```

2. Install WebSphere MQ Internet pass-thru

   The default installation of WebSphere MQ Internet pass-thru sets the ownership of the binaries to `root`, which helps to protect them. Before installation, create an `mqm` group and `mqm` user, as you would for installing WebSphere MQ. After installation, a separate set of directories is created to hold the runtime configuration. The WebSphere MQ Internet pass-thru server will run as the `mqm` account, not as `root`, in order to reduce the exposure if there should be a vulnerability in the code.

   The security for some of the installed directories is relaxed in order to make sample files available. This is not a runtime security exposure because the `/opt` directories are not used for storing the WebSphere MQ Internet pass-thru configuration.

   Example 10-29 on page 163 shows the commands to install WebSphere MQ Internet pass-thru. The commands are run as `root`.

Example 10-29 Install WebSphere MQ Internet pass-thru in default location (/opt/mqipt)

```
rpm -ivh WebSphereMQ-IPT-2.0.0-3.i586.rpm
groupadd mqm
useradd -g mqm -s /bin/bash mqm
mkdir -m 750 -p /var/mqm/scripts
chown -R mqm:mqm /var/mqm
cp mqipt /var/mqm/scripts
chown mqm:mqm /var/mqm/scripts/mqipt
chmod 550 /var/mqm/scripts/mqipt
cp mqipt.init.d /etc/init.d/mqipt
chown root:mqm /etc/init.d/mqipt
chmod 550 /etc/init.d/mqipt
chkconfig --add mqipt
cd /opt
chmod 755 exits ssl web
```

3. Determine certificate DNs for WebSphere MQ Internet pass-thru

   Some of the certificates have already been configured in the gateway queue manager, while the external certificate is newly defined. Example 10-30 shows the DNs of the certificates that must for created for WebSphere MQ Internet pass-thru to use. Each certificate was created for this scenario as a self-signed certificate. The CA that was used to sign certificates for this scenario was implemented with the sha256 signature algorithm. We found that WebSphere MQ Internet pass-thru did not operate correctly with certificates using this algorithm.

   Self-signed certificates must NOT have the isCA flag set. If isCA=yes were present in the certificate, it could be used to sign other certificates. This would create a significant vulnerability in the certificate validation performed by WebSphere MQ Internet pass-thru and WebSphere MQ. This scenario describes using self-signed certificates for all links involving WebSphere MQ Internet pass-thru. If CA-signed certificates are used instead, CRLs must be checked.

Example 10-30 Define certificate DNs that will be required by WebSphere MQ Internet pass-thru

```
CN=Business Partner 1,OU=SA-W216 Residency,O=IBM ITSO,C=US
CN=Business Partner 2,OU=SA-W216 Residency,O=IBM ITSO,C=US
CN=Company 1,OU=SA-W216 Residency,O=IBM ITSO,C=US
```

4. Create the WebSphere MQ Internet pass-thru configuration directories

   Example 10-31 shows the creation of the configuration directory for WebSphere MQ Internet pass-thru. The actual location of the directory is arbitrary. In this case, it is configured to the directory that is set in the **mqipt** start/stop script. The **mqipt** script can be seen in Appendix B, “Additional tooling for WebSphere MQ Internet pass-thru” on page 291.

   The commands shown in Example 10-31 are run as `root`.

Example 10-31 Create the configuration directory for WebSphere MQ Internet pass-thru

```
mkdir /var/mqm/mqipt
mkdir /var/mqm/mqipt/exits
mkdir /var/mqm/mqipt/ssl
mkdir /var/mqm/mqipt/logs
chown -R mqm:mqm /var/mqm/mqipt
chmod -R 700 /var/mqm/mqipt
```

5. Create WebSphere MQ Internet pass-thru certificates

   Example 10-32 shows a process to create a strong random password, a keystore, and a self-signed certificate. The certificate is then extracted from the keystore so that it can be sent to our business partners for addition into their truststores. The certificate is then available to validate our identity. The `Company1.p12` keystore contains the self-signed certificate that will be used for communication with our business partners. The example should be run using the WebSphere MQ Internet pass-thru runtime user ID (`mqm`).

Example 10-32 Create PKCS#12 keystore, private key, and certificate for external links

```
umask 077

cd /var/mqm/mqipt/ssl

# Create a random 64 character string to use for a password
runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65 > Company1.passwd

KEYPW=`cat Company1.passwd`

# Create the key store
runmqakm -fips -keydb -create -db Company1.p12 -pw "$KEYPW" -type pkcs12 -stash
-empty

# Create the self signed certificate
runmqakm -fips -cert -create -db Company1.p12 -stashed -label "mqiptCompany1"
-dn "CN=Company 1,OU=SA-W216 Residency,O=IBM ITSO,C=US" -target Company1.req
-sigalg SHA1WithRSA -size 2048 -expire 396

# Create the certificate file
runmqakm -fips -cert -extract -db Company1.p12 -stashed -file Company1.cer
-label "mqiptCompany1"
```

```
# Generate an obfuscated password file for MQIPT to use
/opt/mqipt/bin/mqiptPW "$KEYPW" Company1.passwd.mqipt

chmod 400 Company1.*
```

Example 10-33 repeats the exercise of creating personal certificates, but these certificates are used for communication to the gateway queue manager. The `BP1.p12` keystore contains the self-signed certificate that is used for BP1 connections to the gateway queue manager. `BP2.p12`, `BP3.p12`, and so on would be used for other business partner connections. Only the creation of the BP1 keystore and certificate is shown. Note the naming. The `BP1` certificate is used to represent `BP1` to the gateway queue manager. It is not used in communication from WebSphere MQ Internet pass-thru to Business Partner 1. The `Company1` certificate (representing our company) is used for all business partner communication. This certificate was created in Example 10-32. The commands that are shown in Example 10-33 should be run as the WebSphere MQ Internet pass-thru runtime user. After the certificate is created, it is extracted to a certificate file so that it can be sent to the MQ administrator of the gateway queue manager.

Example 10-33 Create PKCS#12 keystore, private key, and certificate for links to gateway QM

```
umask 077

cd /var/mqm/mqipt/ssl

# Create a random 64 character password
/opt/mqm/bin/runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65 > BP1.passwd

KEYPW=`cat BP1.passwd`

# Create the key store
/opt/mqm/bin/runmqakm -fips -keydb -create -db BP1.p12 -pw "$KEYPW" -type
pkcs12 -stash -empty

# Create the self-signed certificate
/opt/mqm/bin/runmqakm -fips -cert -create -db BP1.p12 -stashed -label
"mqiptBP1" -dn "CN=Business Partner 1,OU=SA-W216 Residency,O=IBM ITSO,C=US"
-target BP1.req -sigalg SHA1WithRSA -size 2048 -expire 396

# Extract the certificate ready for distribution
runmqakm -fips -cert -extract -db BP1.p12 -stashed -file BP1.cer -label
"mqiptBP1"

# Create the obfuscated password file needed by MQIPT.
/opt/mqipt/bin/mqiptPW "$KEYPW" BP1.passwd.mqipt

chmod 400 BP1.*
```

6. Create WebSphere MQ Internet pass-thru truststores

   MQIPT requires several truststores. There must be a truststore for each business partner, which is used to validate their certificate, and there must be a truststore, which is used to validate the certificate of the gateway queue manager.

Example 10-34 shows the creation of the truststore for validating the gateway queue manager. The same truststore can be used in many places, because it is always being used to validate the certificate of the gateway queue manager. It is called `Company1_trust.p12` because it is used to establish trust with the gateway queue manager, which belongs to Company1.

Separate truststores for each business partner and the gateway queue manager are provided as there is no way to specify a restriction on which certificates within a shared truststore should be trusted by a particular connection. By separating the truststores, the attack profile on each connection is reduced to a single self-signed certificate.

Creating the truststore: The usage of the files is reversed compared to the personal key/certificate stores. The task of creating the truststore is performed by the WebSphere MQ Internet pass-thru runtime account (`mqm`). The certificate added to the truststore is the gateway queue manager self-signed certificate, which was created in Example 10-15 on page 156.

Example 10-34 Create truststore for validating gateway QM certificates

```
# Generate a random 64 character password
/opt/mqm/bin/runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65 > Company1_trust.passwd

KEYPW=`cat Company1_trust.passwd`

# Create the trust store
/opt/mqm/bin/runmqakm -fips -keydb -create -db Company1_trust.p12 -pw "$KEYPW"
-type pkcs12 -stash -empty

# Add the trusted certificate to the trust store
/opt/mqm/bin/runmqakm -fips -cert -add -db Company1_trust.p12 -stashed -type
pkcs12 -label "QMGW1 self-signed" -file QMGW1.cer -format ascii -trust enable

# Create the obfuscated password files needed by MQIPT
/opt/mqipt/bin/mqiptPW "$KEYPW" Company1_trust.passwd.mqipt
```

Shown in Example 10-35, `BP1_trust.p12` is the truststore that is used to validate certificates that are presented by Business Partner 1. Business Partner 1 will be using a self-signed certificate, so there is only one certificate in the store. Truststores for other business partners should be created using similar commands. The business partner has to send you the self-signed certificate that they have created. For further information, see “Create the WebSphere MQ Internet pass-thru configuration file” on page 167.

Example 10-35 Create truststore for validating Business Partner 1 certificates

```
# Generate a random 64 character password
/opt/mqm/bin/runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65 > BP1_trust.passwd

KEYPW=`cat BP1_trust.passwd`

# Create the trust store
/opt/mqm/bin/runmqakm -fips -keydb -create -db BP1_trust.p12 -pw "$KEYPW" -type
pkcs12 -stash -empty

# Add the trusted certificate into the trust store
```

```
/opt/mqm/bin/runmqakm -fips -cert -add -db BP1_trust.p12 -stashed -type pkcs12
-label "Business Partner 1 self signed" -file partner1.cer -format ascii -trust
enable

# Generate the obscured password file needed by MQIPT.
/opt/mqipt/bin/mqiptPW "$KEYPW" BP1_trust.passwd.mqipt
```

After completing work with the certificate stores, all files in the `/opt/mqm/mqipt/ssl` directory should be set to read-only to the owner, and no access to others, which is shown in Example 10-36.

Example 10-36 Set all WebSphere MQ Internet pass-thru key and certificate files to read-only

```
chmod 400 /var/mqm/mqipt/ssl/*
```

This section has shown usage of self-signed certificates and passwords held in files as clear text. See “Create the WebSphere MQ Internet pass-thru configuration file” and “Obscuring password files” for further discussion on issues associated with these uses.

7. Create the WebSphere MQ Internet pass-thru configuration file

   The WebSphere MQ Internet pass-thru configuration file is called `mqipt.conf`. It is located in the configuration directory, which is named when the mqipt service is started. The example scripts for this scenario assume that the configuration directory location is `/var/mqm/mqipt`.

   Example 10-37 shows the Global section of the `mqipt.conf` file that was created for this scenario. The Route section is then shown later.

Example 10-37 Global section of the mqipt.conf configuration file

```
##############################################################################
#
#
# Global default properties for all routes
#
# AccessPw, CommandPort, ConnectionLog, RemoteShutDown and MaxLogFileSize can
only
# be used in the global section.
#
[global]
# Global only parameter values
AccessPW=WeakPassword
#CommandPort=1881
RemoteShutDown=false
ConnectionLog=true
MaxLogFileSize=1024
SecurityManager=true
SecurityManagerPolicy=/var/mqm/mqipt/mqiptSecurity.policy
# Default values for route definitions
MinConnectionThreads=5
MaxConnectionThreads=100
IdleTimeout=20
ClientAccess=false
QMgrAccess=true
HTTP=false
HTTPChunking=false
Trace=0
```

The AccessPW is set in the file, even though it is a weak password and is in clear text. There is no way within WebSphere MQ Internet pass-thru or the Java security policy to prevent outside users from running a client program to view and change the WebSphere MQ Internet pass-thru configuration if the CommandPort is open and no password is set. It is better to define a password so there is some protection if a CommandPort is defined.

There is currently no way to bind the CommandPort listener to a specific network interface or address. The CommandPort does not receive encrypted credentials from the client application. A firewall should be configured so that outside servers do not have access to the CommandPort. In order to enforce the strongest available security controls, this configuration removes the CommandPort statement. As well as blocking access from the GUI administration client, normal stop, restart, or reload operations via the **mqiptAdmin** command-line tool cannot be used. The supplied **mqipt** script includes a "clean" option that allows shutdown even if the CommandPort is disabled. With the CommandPort disabled, a configuration change that is made to the `mqipt.conf` file requires that the WebSphere MQ Internet pass-thru service must be stopped and restarted to activate the new configuration.

After `mqipt.conf` is created, set the permissions to 600 until all information has been added. After the configuration is complete and tested, set all files in the configuration directory to mode 400 (Only read allowed to `mqm`, no other access permitted).

8. Update the Java virtual machine (JVM) security policy files

   WebSphere MQ Internet pass-thru should be configured to use a private security policy file. This file is secured in the WebSphere MQ Internet pass-thru configuration directory, together with the rest of the WebSphere MQ Internet pass-thru configuration. The JVM that is used to run WebSphere MQ Internet pass-thru needs to be configured to allow this custom policy file to be used.

   The commands in Example 10-38 show how to use the **mqipt** script **jre** option to find the JRE that WebSphere MQ Internet pass-thru will use. It then shows the update to the existing security files (`java.security` and `java.policy`) to enable custom policies for the JVM.

Example 10-38 Update the WebSphere MQ JVM

```
/opt/mqm/scripts/mqipt jre
# Now change to the directory where the java program is located, as
# returned by the previous command.
cd /opt/mqm/java/jre64/jre/
# Edit the java.security file, and add the policy to allow policy configuration
# if it is not already allowed.
vi lib/security/java.security
   # whether or not we allow an extra policy to be passed on the command line
   # with -Djava.security.policy=somefile. Comment out this line to disable
   # this feature.
   policy.allowSystemProperty=true

# Edit the java.policy file so that MQIPT has getPolicy privilege
vi lib/security/java.policy

   // Allow MQIPT to use specific java policy control

   grant codeBase "file:///opt/mqipt/lib/com.ibm.mq.ipt.jar" {
      permission java.security.SecurityPermission "getPolicy";
   };
```

9. Create the WebSphere MQ Internet pass-thru security policy file

   WebSphere MQ Internet pass-thru includes a sample policy file in the `/opt/mqipt/ssl` directory. This file can be copied to the configuration directory and changed to reflect the installation and configuration directories that are used. Permissions for the routes and command server ports would then be added. However, the supplied sample policy is targeted for Windows and would need to be substantially edited to be useful.

   In Example 10-39, a useful Linux environment policy file is shown. If you are going to implement on a UNIX style platform, use this file instead.

   In this example policy file, permission for the WebSphere MQ Internet pass-thru service to use the CommandPort (default 1881) is not included, so even if a CommandPort is included in the `mqipt.conf` file, it will not be used. The additional permissions that are needed to enable the CommandPort using port 1881 are shown in Example 10-40 on page 170.

Example 10-39 Update the sample Java security policy file to apply to this scenario

```
cd /var/mqm/mqipt

# create the new policy file
touch ./mqiptSecurity.policy

# set permissions so only the owner can see or update the file
chmod 600 mqiptSecurity.policy

# Populate the file with required initial values needed for all
# MQIPT systems
cat <<POLICY > mqiptSecurity.policy
grant codeBase "file:///opt/mqipt/lib/com.ibm.mq.ipt.jar" {
  permission java.util.PropertyPermission "user.dir", "read";
  permission java.util.PropertyPermission "user.home", "read";
  permission java.util.PropertyPermission "file.encoding", "read";
  permission java.util.PropertyPermission "java.version", "read";
  permission java.util.PropertyPermission "os.name", "read";
  permission java.io.FilePermission "/opt/mqipt", "read";
  permission java.io.FilePermission "/var/mqm/mqipt/errors/*", "read, write";
  permission java.io.FilePermission "/var/mqm/mqipt/logs/*", "read, write";
  permission java.io.FilePermission "/var/mqm/mqipt/exits/-", "read";
  permission java.io.FilePermission "/var/mqm/mqipt/ssl/*", "read";
  permission java.io.FilePermission "/opt/mqipt/lib/*", "execute";
  permission java.io.FilePermission "/opt/mqipt/bin/*", "execute";
  permission java.io.FilePermission "/var/mqm/mqipt/*", "read, write";
  permission java.lang.RuntimePermission "setSecurityManager";
};
POLICY
```

Example 10-40 shows the Java policy permissions that are needed to enable WebSphere MQ Internet pass-thru to open a command port. In a production environment, these permissions should not be granted because there is no way to control the source of commands reaching WebSphere MQ Internet pass-thru. The Java policy manager is designed to protect the system from the Java application, not the Java application from the rest of the world.

The accept permission allows WebSphere MQ Internet pass-thru to accept connections from an ephemeral source port on the local machine, using address 127.0.0.1. If there were no routes enabled, this would mean that the CommandPort could only be accessed from the server hosting WebSphere MQ Internet pass-thru.

When a route is added and the Java policy is updated to allow connection from another server, as described in “Update Java security policy to include routes” on page 172, additional accept permissions are granted. Although they are intended to allow those servers to connect to the ports for the route, this is not enforced by the Java policy manager. Those servers can now connect to any port on which the WebSphere MQ Internet pass-thru application is listening.

During the development of this scenario, the only way that was found to protect access to the CommandPort was to enforce controls by using firewalls. Because these controls are implemented outside of the products that the WebSphere MQ administrator manages, we chose to disable the CommandPort entirely, rather than trust that access to it would be controlled elsewhere.

Example 10-40 Java policy permissions to enable WebSphere MQIPT to open a command port

```
permission java.net.SocketPermission "localhost:1881", "listen";
permission java.net.SocketPermission "localhost:1025-", "accept";
```

10. Create the route to Business Partner 1 from the gateway queue manager

Example 10-41 on page 170 shows the definitions to enable SSL connections from our gateway queue manager to Business Partner 1. It references the certificate, trust, and password files that were created previously. The SSLServer values are used to communicate with the gateway queue manager and the SSLClient values are used to connect to Business Partner 1. Add these lines to the `mqipt.conf` file that was created in Example 10-37 on page 167.

Example 10-41 Route from gateway queue manager to Business Partner 1

```
[route]
Name=GW1toBP1
Active=true
ListenerPort=5001
Destination=9.42.171.232
DestinationPort=14142
LocalAddress=9.42.170.181
OutgoingPort=0
Trace=0
SSLServer=true
SSLServerKeyRing=/var/mqm/mqipt/ssl/BP1.p12
SSLServerKeyRingPW=/var/mqm/mqipt/ssl/BP1.passwd
SSLServerCAKeyRing=/var/mqm/mqipt/ssl/Company1_trust.p12
SSLServerCAKeyRingPW=/var/mqm/mqipt/ssl/Company1_trust.passwd
SSLServerAskClientAuth=true
SSLServerCipherSuites=SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSLServerSiteLabel=mqiptBP1
SSLServerDN_CN=QMGW1
SSLServerDN_OU=SA-W216 Residency
SSLServerDN_O=IBM ITSO
SSLServerDN_C=US
SSLClient=true
SSLClientKeyRing=/var/mqm/mqipt/ssl/Company1.p12
SSLClientKeyRingPW=/var/mqm/mqipt/ssl/Company1.passwd
```

```
SSLClientCAKeyRing=/var/mqm/mqipt/ssl/BP1_trust.p12
SSLClientCAKeyRingPW=/var/mqm/mqipt/ssl/BP1_trust.passwd
SSLClientAskClientAuth=true
SSLClientCipherSuites=SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSLClientSiteLabel=mqiptCompany1
SSLClientDN_CN=Business Partner 1
SSLClientDN_OU=SA-W216 Residency
SSLClientDN_O=IBM ITSO
SSLClientDN_C=US
LDAP=false
```

11. Create the route from Business Partner 1 to the gateway queue manager

    Example 10-42 shows the additional configuration statements to define the return channel from Business Partner 1 to the gateway queue manager. This example is the inverse of the route that is defined in Example 10-41. LDAP information is not included because the scenario used self-signed certificates, which cannot be revoked. The outbound connection is to the external interface address on the gateway queue manager server.

Example 10-42 Route from Business Partner 1 to gateway queue manager

```
[route]
Name=BP1toGW1
Active=true
ListenerPort=6001
Destination=192.168.103.104
DestinationPort=1416
LocalAddress=192.168.103.61
OutgoingPort=0
Trace=0
SSLServer=true
SSLServerKeyRing=/var/mqm/mqipt/ssl/Company1.p12
SSLServerKeyRingPW=/var/mqm/mqipt/ssl/Company1.passwd
SSLServerCAKeyRing=/var/mqm/mqipt/ssl/BP1_trust.p12
SSLServerCAKeyRingPW=/var/mqm/mqipt/ssl/BP1_trsut.passwd
SSLServerAskClientAuth=true
SSLServerCipherSuites=SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSLServerSiteLabel=mqiptCompany1
SSLServerDN_CN=Business Partner 1
SSLServerDN_OU=SA-W216 Residency
SSLServerDN_O=IBM ITSO
SSLServerDN_C=US
SSLClient=true
SSLClientKeyRing=/var/mqm/mqipt/ssl/BP1.p12
SSLClientKeyRingPW=/var/mqm/mqipt/ssl/BP1.passwd
SSLClientCAKeyRing=/var/mqm/mqipt/ssl/Company1_trust.p12
SSLClientCAKeyRingPW=/var/mqm/mqipt/ssl/Company1_trust.passwd
SSLClientAskClientAuth=true
SSLClientCipherSuites=SSL_RSA_WITH_3DES_EDE_CBC_SHA
SSLClientSiteLabel=mqiptBP1
SSLClientDN_CN=QMGW1
SSLClientDN_OU=SA-W216 Residency
SSLClientDN_O=IBM ITSO
SSLClientDN_C=US
LDAP=false
```

12. Update Java security policy to include routes

Three security policy statements are needed to enable WebSphere MQ Internet pass-thru to complete a connection between two queue managers. The statements are similar, whether the connection is from the gateway queue manager to the business partner, or from the business partner to the gateway. Example 10-43 shows the permissions that are granted for this scenario for the routes that are shown in Example 10-41 on page 170 and Example 10-42 on page 171. The statements shown were extracted from the `mqiptSecurity.policy` file, but were created by using the **`policytool`** as explained in "Updating the policy file using policytool" on page 173.

The following security policy statements are required:

- Allow WebSphere MQ Internet pass-thru to listen on the port specified for the connection.
- Allow WebSphere MQ Internet pass-thru to accept a connection request from one partner.
- Allow WebSphere MQ Internet pass-thru to open a connection (connect) to the other partner.

Example 10-43 Permissions that are granted for this scenario

```
permission java.net.SocketPermission "*:5001", "listen";
permission java.net.SocketPermission "192.168.103.104:1025-", "accept";
permission java.net.SocketPermission "9.42.171.232:14142", "connect, resolve";

permission java.net.SocketPermission "*:6001", "listen";
permission java.net.SocketPermission "9.42.171.232:1025-", "accept";
permission java.net.SocketPermission "192.168.103.104:1416", "connect, resolve";
```

13. Add the WebSphere MQ Internet pass-thru security exit to control source address to port usage

The Java security policy alone is not able to restrict access by a gateway queue manager or business partner queue manager to a specific listener port opened by WebSphere MQ Internet pass-thru.

The gateway queue manager controls connections from WebSphere MQ Internet pass-thru by checking the certificate DN using SSLPEER. WebSphere MQ Internet pass-thru assigns the certificate to a route based on the incoming port number. It is therefore important to control which business partner can use which port.

Based on the principle of *defense in depth*, there are multiple tools that limit access to the incoming ports:

- Firewall rules
- SSL DN validation
- WebSphere MQ Internet pass-thru security exit

The sample firewall rule is shown in 10.3.5, "Firewall configuration" on page 178.

A security exit is required to enable the checking of the source IP address against the port number. A sample exit (`itsoBlockExit`) is included as an example of how this could be done. A local firewall on the machine hosting WebSphere MQ Internet pass-thru could be used to restrict access to the correct ports. This firewall could be a Windows firewall or software, such as iptables or ipchains on a GNU Linux OS.

For information about the exit that is configured in this scenario, see Appendix B, "Additional tooling for WebSphere MQ Internet pass-thru" on page 291. The appendix contains the source, compilation instructions, and a discussion of the functions that are performed by the exit.

Example 10-44 shows the exit configuration file `itsoBlockExit.conf`, which is located in the exit directory (`/var/mqm/mqipt/exits`). It includes validation rules for both the incoming and outgoing paths. Create this file with the rules shown.

Example 10-44 itsoBlockExit.conf

```
5001 192.168.103.104
6001 9.42.171.232
```

Example 10-45 shows the additional lines that are needed in the route configuration sections for the BP1toGW1 and GW1toBP1 routes. If all routes are required to validate using the same exit, this could be placed in the [global] section of the `mqipt.conf` file.

Example 10-45 Additional lines in mqipt.conf routes to enable the itso BlockExit

```
SecurityExit=true
SecurityExitName=itsoBlockExit
SecurityExitPath=/var/mqm/mqipt/exits
SecurityExitTimeout=10
```

14. Add the security policy to permit exit execution

    Example 10-46 on page 173 shows the additional Java security policy permissions that are needed so that WebSphere MQ Internet pass-thru can execute the itsoBlockExit (or any other exit). This line needs to be added to the `/var/mqm/mqipt/mqiptSecurity.policy` file that is referenced in the `mqipt.conf`, using the **`policytool`**.

    Example 10-46 Java security policy permissions to execute itsoBlockExit

    ```
    permission java.lang.RuntimePermission "createClassLoader";
    ```

Updating the policy file using policytool

Follow these steps to update the policy file using the **`policytool`** utility:

1. Start **`policytool`**. The commands to start **`policytool`** are shown in Example 10-47. Figure 10-5 shows the initial panel that is presented when **`policytool`** is run.

   Example 10-47 Commands to start policytool to edit the WebSphere MQIPT policy file

   ```
   # change to the mqipt configuration directory
   cd /var/mqm/mqipt

   # start policy tool to edit the mqiptSecurity.policy file
   policytool -file mqiptSecurity.policy .
   ```

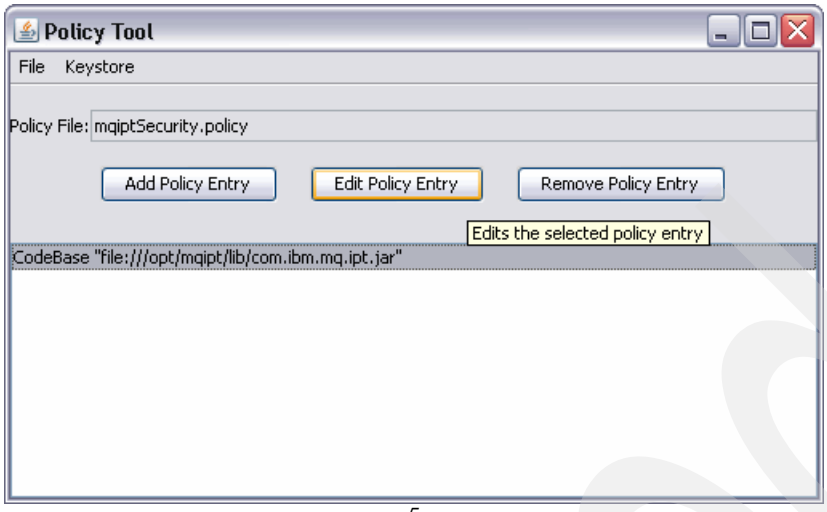


Figure 10-5 Policy Tool initial window[5]

2. Select the CodeBase entry for WebSphere MQ Internet pass-thru and click **Edit Policy Entry.** The Policy Entry panel is shown in Figure 10-6 on page 174.

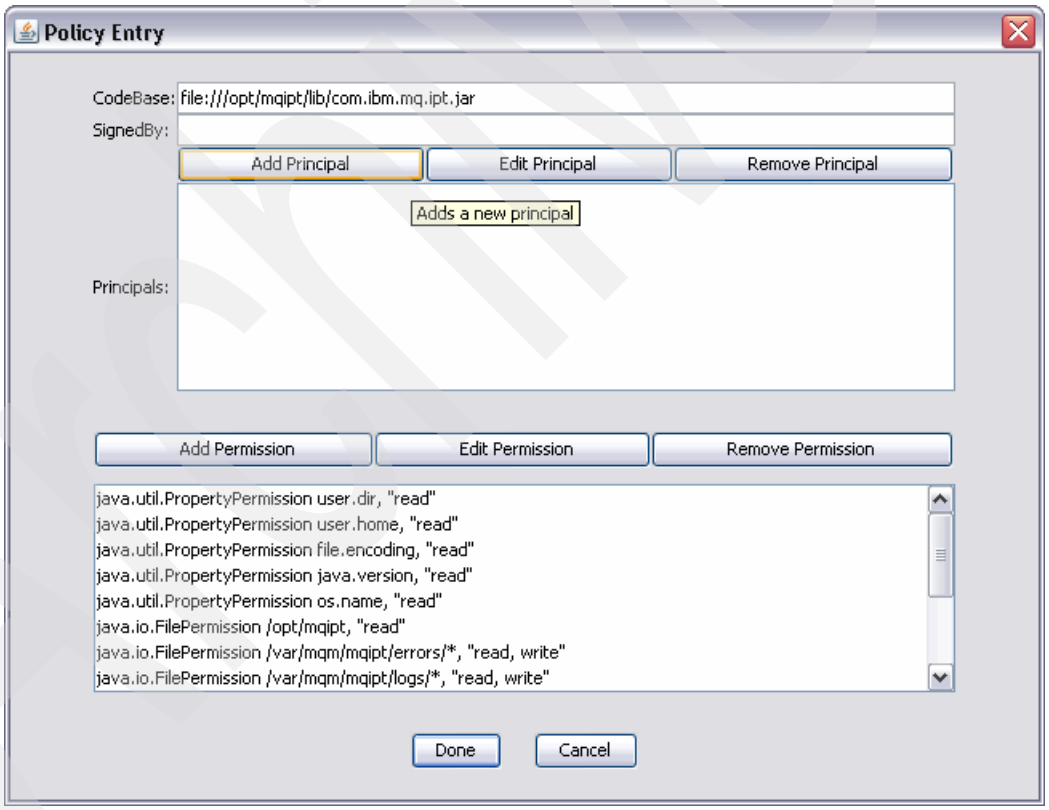


Figure 10-6 Policy Entry panel

3. Click **Add Permission**. This option will display the Permissions panel as illustrated in Figure 10-7. The Permissions panel is then used to create the permission entry, which is added to the `mqiptSecurity.policy` file.

[5] Note: The versions of policytool supplied with different JREs may have different appearances, but the capabilities that are provided will be equivalent to those capabilities that are shown in the figures.

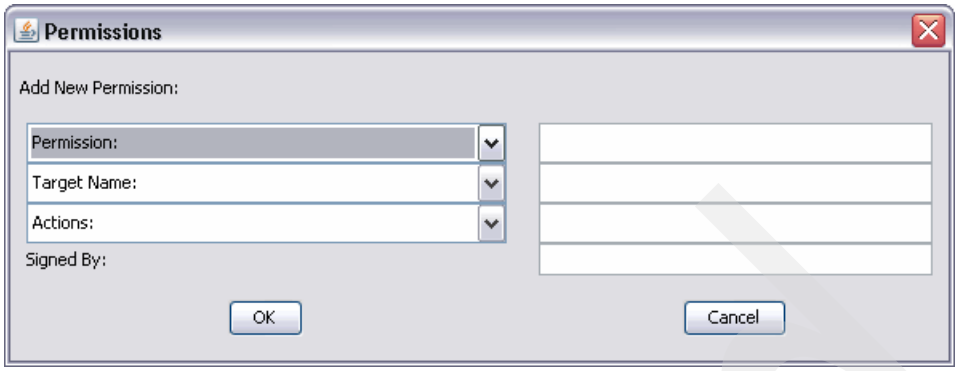


Figure 10-7 Permissions entry panel

4. Select **Permission**, add the **Target Name**, and select **Actions**, then click **OK**. The window in Figure 10-8 on page 175 opens, granting the listen SocketPermission on port 5001.

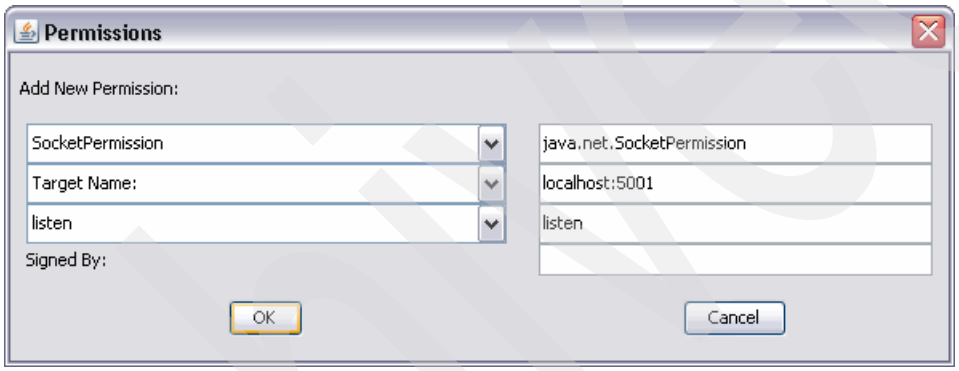


Figure 10-8 Add a policy permission using policytool

5. You can click Add, Edit, or Remove to work with any other policy permissions that are needed by selecting options on the Policy Entry panel. See Figure 10-6 on page 174.
6. Click **Done**, and then select **File** → **Save**.

Additional business partners

There are two ways that could be used to add extra business partners to this configuration:

- Create a totally separate WebSphere MQ Internet pass-thru configuration on this server or another server, and run it in parallel. To create a new WebSphere MQ Internet pass-thru instance on the same server, a separate configuration directory is needed. Then, repeat all the steps to implement WebSphere MQ Internet pass-thru except for the installation.
- Implement a pair of routes within the existing WebSphere MQ Internet pass-thru server. The steps that are associated with creating business partner routes are repeated for each extra business partner.

The advantage of using a single WebSphere MQ Internet pass-thru instance for multiple business partners is that it simplifies the infrastructure that has to be built and managed. However, an exit is required in order to prevent one partner from connecting to the port reserved for a different partner. This attempt should be blocked at the firewall, and also by SSL DN checking, but by using the security principle of *defense in depth*, an exit provides an additional layer of protection.

Using separate WebSphere MQ Internet pass-thru instances for each business partner means that the Java security policy is able to restrict connections to only those connections from a valid partner, and it means that the configuration for each WebSphere MQ Internet pass-thru can be changed without affecting any other routes.

Trusting self-signed certificates

Organizations trust CA-signed certificates because they accept that the CA (a trusted third party) has validated the certificate request before signing it.

When a self-signed certificate is used instead, the companies exchanging the self-signed certificates have to verify the validity of the business partner and the validity of the certificate that is received. It is important to verify that a certificate has not been intercepted and replaced in transit. An undetected certificate replacement would facilitate a *man in the middle* attack on the traffic between the companies.

In order to validate a received self-signed certificate, you have to have established a trusted communications channel with the business partner or WebSphere MQ administrator. This could be as simple as having met them, and recognizing their voice.

After the certificate is received and before adding it to the truststore, use a tool, such as Windows Crypto Shell extensions (the default application on Windows operating systems for handling `.cer`, `.crt`, `.der`, and `.p7b` files), **`openssl`**, or another X.509-compliant tool on Linux to examine the certificate.

Confirm the fingerprint or thumbprint (depending on how you displayed the certificate) against the value that is provided through a different communication path with the business partner. For example, if you receive the certificate via email, you could use a phone call to verify the fingerprint or thumbprint.

The important thing is not the mechanism for delivery and validation of certificates, but that the certificates are validated using a different path before being trusted.

Obscuring password files

Obscuring the content of the password files for the certificates and truststores does not significantly protect them because the encryption used is easily reversible. Strong protection at the OS level of all files that are used for WebSphere MQ Internet pass-thru is the only way to ensure security of the cryptographic keys. Exposure of the keys means that the security of any link using them is compromised.

Because of this, the clear text password files used when the PKCS#12 key and certificate stores were created are not deleted from the file system (there is no point). The WebSphere MQ Internet pass-thru tool to encrypt the password files (mqiptPW) is used to create separate files, which are named in the WebSphere MQ Internet pass-thru configuration. These steps were shown in the examples where the various key and truststores were created (Example 10-32 on page 164, Example 10-33 on page 165, Example 10-34 on page 166, and Example 10-35 on page 166).

The one advantage of obfuscated passwords over clear text passwords is that they may make it more difficult for someone looking over a shoulder ("*shoulder surfing*") to recall the password to form part of a later attack. This is addressed in this scenario by generating a random 64-character string to use for the passwords.

10.3.4 Application configuration

The focus of the scenarios in this book is the administrative controls and configuration needed to implement WebSphere MQ in a way that provides authentication of users and authorization of actions. In most instances, the details of the application are not significant.

However, some of the configuration in this scenario provided definitions that should be used by an application to hide its location from a partner. Rather than write programs to illustrate these features, the q program, which is supplied in WebSphere MQ SupportPac ma01, is used. This program is available from the SupportPac website:

http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg27007197#1

The program is configurable using arguments on the shell execution, providing a clear indication of what needs to be done, without requiring the development of code.

Example 10-48 on page 177 shows the SSL and channel table set-up to enable an application program (in this case, the q program from ma01) to send a request message to the BP1 queue manager at Business Partner 1. It does the following actions:

- Copies the channel table from the queue manager directory to the application directory
- Creates a keystore and personal certificate for the application
- Creates an `mqclient.ini` file to configure client options
- Sets up environment variables so that these things are available to the running application

The tests were run with the user name "`casey`".

Example 10-48 Set up the environment to send messages using a client application

```
cd /home/casey/.mq

cp /var/mqm/qmgrs/QM1/\@ipcc/AMQCLCHL.TAB ./B2BCHL.TAB

runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
'\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65 > mqcust1.passwd

runmqakm -fips -keydb mqcust1 -create -db mqcust1.kdb -pw "`cat mqcust1.passwd`"
-type cms -stash -empty

runmqakm -fips -cert -add -db mqcust1.kdb -stashed -file wmqca1.cer -label "WMQCA1
MQ Root CA" -trust enable

runmqakm -fips -cert -add -db mqcust1.kdb -stashed -file wmqca2.cer -label "WMQCA2
MQ Intermediate CA" -trust enable

runmqakm -fips -certreq -create -db mqcust1.kdb -stashed -label
ibmwebspheremqcasey -dn "cn=Requester Application 1,OU=SA-W216 Residency,O=IBM
ITSO,C=US" -target mqcust1.req -sigalg sha256 -size 2048

runmqakm -fips -cert -receive -db mqcust1.kdb -stashed -file mqcust1.cer

echo "Test B2B message from Company1 to Business Partner 1" > onemessage.txt

echo "ClientExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64
SSL:
   OCSPCheckExtensions=YES
```

```
   SSLFipsRequired=YES
" > mqclient.ini


export MQCHLLIB=/home/casey/.mq
export MQCHLTAB=B2BCHL.TAB
export MQSSLKEYR=/home/casey/.mq/mqcust1
export MQSSLFIPS=YES
export MQCLNTCF="file:///home/casey/.mq/mqclient.ini"
```

Example 10-49 shows the commands to send a message (from a file) to the business partner request queue with various options. The first message is sent using a ReplyToQ alias. The second is sent using a normal ReplyToQ. In both cases, report options are specified. They will be removed by the channel exit implemented at the gateway queue manager. (For information about the itsoME exit, see Appendix A, “Working with the itsoME message exit” on page 263.)

Example 10-49 Send request messages from Application 1 to Business Partner 1

```
q -t -z -anR -n"cafd cdfd efd xfd" -f onemessage.txt -j MQSTR -l mqic -m \*App1QM1
-o BP.APP1.REQUEST -r BP.APP1.RESPONSE.VIA.GW

q -t -z -anR -n"cafd cdfd efd xfd" -f onemessage.txt -j MQSTR -l mqic -m \*App1QM1
-o BP.APP1.REQUEST -r BP.APP1.RESPONSE
```

The messages that are sent by Example 10-49 on page 178 are requests, and the business partner application should send a response. To emulate that behavior, the amqsech0 sample program was used at the BP1 queue manager to generate a duplicate of the request message and send it back to the reply queue. Due to the aliasing performed either by the requesting application or by the itsoMQ exit, the target of the reply is the queue manager alias “COMPANY1”. The business partner queue manager is implemented so that these messages are sent to the gateway queue manager operated by Company1. This has definitions of the target queues so that messages are forwarded to the application destination.

Output from sample program amqsbcg in Example 10-52 on page 180, Example 10-53 on page 181 and Example 10-54 on page 182 show the request message and the resultant message in the reply queue, with correlation IDs matching the original request message.

10.3.5 Firewall configuration

The expected environment for a B2B WebSphere MQ scenario has four network zones, which are separated by three firewalls as illustrated in Figure 10-2 on page 145. The zones are the internal application zone (tier 3), gateway zone (tier 2), external connectivity zone (tier 1), and outside our organization (tier 0). Firewalls between each zone are configured to only permit the specific connections required (address and port) to enable the B2B connectivity illustrated.

The rules for this scenario are described in Table 10-1.

Table 10-1 Source port designation

Name	Source address	Source port	Destination address	Destination port
App-GW	192.168.102.101	1025-[a]	192.168.102.104	1415
GW-App	192.168.102.104	1025-	192.168.102.101	1414
GW-MQIPT	192.168.103.104	1025-	192.168.103.61	5001
MQIPT-GW	192.168.103.61	1025-	192.168.103.104	1416
MQIPT-BP1	9.42.170.181	1025-	9.42.171.232	14142
BP1-MQIPT	9.42.171.232	1025-	9.42.170.181	6001

a. The hyphen (-) designates this port and all ports numbered above it.

These rules would be applied across the three separate firewalls.

10.4 Results of testing

After setting up the environment, test messages were generated using the ma01 q program. The messages were captured at various points, and the amqsbcg0 sample program was used to display the message content and descriptor. The output from these programs and the debug log of the itsoME exit show how messages flow between queue managers and how they are modified.

Example 10-50 and Example 10-51 show the debug log from the gateway queue manager itsoME exit. The debug option is defined at compile time, so it is normally disabled. A recompilation is required in order to activate debug mode. Debug mode was used during development of the exit so that the actions taken by the exit could be illustrated.

Example 10-50 Debug log of message where replytoq alias was used to generate correct reply details

```
itsoME: ver=1.00 ExitId=MQXT_CHANNEL_MSG_EXIT (12) ExitReason=MQRX_MSG (13)
ChannelType=MQCHT_RECEIVER (3)
itsoME: QMgrName="QMGW1"
itsoME: ChannelName="QM1.QMGW1"
itsoME: msg MsgType=1 Report=0x07e03f00
itsoME: Unwanted Report options was removed (EXC/EXP).
itsoME: Unwanted Report options was removed (coa/cod).
itsoME: Replaced UserIdentifier with "mcaqm1      "
itsoME: exit ExitResponse=MQXCC_OK (0)
```

Example 10-51 Debug log of a message where the exit repairs the replyto details

```
itsoME: ver=1.00 ExitId=MQXT_CHANNEL_MSG_EXIT (12) ExitReason=MQRX_MSG (13)
ChannelType=MQCHT_RECEIVER (3)
itsoME: QMgrName="QMGW1"
itsoME: ChannelName="QM1.QMGW1"
itsoME: msg MsgType=1 Report=0x07e03f00
itsoME: Unwanted Report options was removed (EXC/EXP).
itsoME: Unwanted Report options was removed (coa/cod).
itsoME: Replaced UserIdentifier with "mcaqm1      "
```

```
itsoME: Replaced WQmgr "COMPANY1" Qmgr1"QM1"
itsoME: Old ReplyToQ "BP.APP1.RESPONSE"
itsoME: new ReplyToQ "QM1.BP.APP1.RESPONSE"
itsoME: Replaced ReplyToQMgr with "COMPANY1"
itsoME: Replaced ReplyToQueue with "QM1.BP.APP1.RESPONSE"
itsoME: exit ExitResponse=MQXCC_OK (0)
```

Example 10-52 shows a dump of the message sent to Business Partner 1. The amqsbcg sample program was used to produce this output. The message has been altered from the message that was sent by the application. The altered values have been highlighted. This message was sent from an application that did not correctly set the reply destination using a replyToQ alias.

The following values are altered:

- The context user ID has been replaced by the gateway queue manager MCAUSER ID.
- The report options requesting COA/COD/Expiry notification, and so on, have been removed.
- The ReplyToQ and ReplyToQM values have been changed to provide a path back to the gateway instead of to the originating queue manager (QM1).

Example 10-52 amqsbcg message print of the message that is transmitted to Business Partner 1

```
AMQSBCG0 - starts here
**********************

 MQOPEN - 'COMPANY1.REQUESTS'


 MQGET of message number 1, CompCode:0 Reason:0
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 1
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
  Format : 'MQSTR   '
  Priority : 0  Persistence : 0
  MsgId : X'414D5120514D3120202020202020201332F34F04080420'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : 'QM1.BP.APP1.RESPONSE                          '
  ReplyToQMgr    : 'COMPANY1                                      '
  ** Identity Context
  UserIdentifier : 'mcaqm1      '
  AccountingToken :
   X'0335313700000000000000000000000000000000000000000000000000000000000006'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : 'q                            '
  PutDate  : '20120715'    PutTime  : '19102830'
  ApplOriginData : '    '

  GroupId : X'000000000000000000000000000000000000000000000000'
```

```
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'

****   Message      ****

 length - 38 of 38 bytes

00000000:  5265 7175 6573 7420 4D65 7373 6167 6520          'Request Message '
00000010:  6672 6F6D 204E 6569 6C20 4361 7365 7920          'from Neil Casey '
00000020:  746F 2042 5031                                   'to BP1          '



 No more messages
 MQCLOSE
 MQDISC
```

Example 10-53 shows the original message that was created by the sending application before it was modified by the gateway queue manager message exit (itsoME).

Example 10-53 amqsbcg dump of the original message that was sent by Application 1

```
AMQSBCG0 - starts here
**********************

 MQOPEN - 'BP.APP1.REQUEST.LOCAL'


 MQGET of message number 1, CompCode:0 Reason:0
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 132136704  MsgType : 1
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
  Format : 'MQSTR   '
  Priority : 0  Persistence : 0
  MsgId : X'414D5120514D3120202020202020201332F34F04080420'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : 'BP.APP1.RESPONSE                            '
  ReplyToQMgr    : 'QM1                                         '
  ** Identity Context
  UserIdentifier : 'mcaapp1     '
  AccountingToken :
   X'03353137000000000000000000000000000000000000000000000000000000000006'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : 'q'
  PutDate  : '20120715'    PutTime  : '19102830'
  ApplOriginData : '    '

  GroupId : X'000000000000000000000000000000000000000000000000'
```

```
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'

****   Message      ****

 length - 38 of 38 bytes

00000000:  5265 7175 6573 7420 4D65 7373 6167 6520          'Request Message '
00000010:  6672 6F6D 204E 6569 6C20 4361 7365 7920          'from Neil Casey '
00000020:  746F 2042 5031                                   'to BP1          '



 No more messages
 MQCLOSE
 MQDISC
```

Example 10-54 shows the response message that was sent by the business partner application. The message type is 2 (reply) and the CorrelID matches the MessageID of the original request. Also, the ReplyToQMgr was replaced by the itsoME exit as part of its processing of all messages from this business partner.

Example 10-54 Response message returned by Business Partner

```
AMQSBCG0 - starts here
**********************

 MQOPEN - 'BP.APP1.RESPONSE'


 MQGET of message number 1, CompCode:0 Reason:0
****Message descriptor****

  StrucId  : 'MD  '  Version : 2
  Report   : 0  MsgType : 2
  Expiry   : -1  Feedback : 0
  Encoding : 546  CodedCharSetId : 1208
  Format : 'MQSTR   '
  Priority : 0  Persistence : 0
  MsgId : X'414D5120425031202020202020202020207D89FF4F02560020'
  CorrelId : X'414D5120514D3120202020202020202020201332F34F04080420'
  BackoutCount : 0
  ReplyToQ       : 'COMPANY1.QM1.BP.APP1.RESPONSE                   '
  ReplyToQMgr    : 'LOCALQM                                         '
  ** Identity Context
  UserIdentifier : 'mcacust1    '
  AccountingToken :
   X'0335303100000000000000000000000000000000000000000000000000000006'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : 'amqsech                      '
  PutDate  : '20120715'    PutTime  : '19145117'
  ApplOriginData : '    '
```

```
  GroupId : X'000000000000000000000000000000000000000000000000'
  MsgSeqNumber   : '1'
  Offset         : '0'
  MsgFlags       : '0'
  OriginalLength : '-1'

****   Message      ****

 length - 38 of 38 bytes

00000000:  5265 7175 6573 7420 4D65 7373 6167 6520          'Request Message '
00000010:  6672 6F6D 204E 6569 6C20 4361 7365 7920          'from Neil Casey '
00000020:  746F 2042 5031                                   'to BP1          '



 No more messages
 MQCLOSE
 MQDISC
```

The second application test scenario (BP1 sends a request message to Application 2 in Company1) was also tested, but since it is just the reverse of the Application 1 actions described above, there is nothing interesting to illustrate in the messages or logs produced.

10.5 Advanced Encryption Standard support in WebSphere MQ Internet pass-thru

Although the current version (v2.0.0.3) of WebSphere MQ Internet pass-thru supports a number of cipher suites using Advanced Encryption Standard (AES) encryption, we found during the development of this scenario that they did not interoperate with WebSphere MQ 7.5. WebSphere MQ uses TLS 1.0 or TLS 1.2 rather than SSL 3.0 when negotiating AES cipher suites.

APAR IV25345 will address this issue by updating the IBM Secure Socket Lite (SSLite) package that is used by WebSphere MQ Internet pass-thru. In order to take advantage of the performance and encryption strength advantages of the AES cipher suites over the much older triple DES suite (such as SSL_RSA_WITH_3DES_EDE_CBC_SHA), you should track the APAR and install updates from it when it becomes available.

Search for and track the APAR at this website:

https://www-304.ibm.com/ibmlink/sis/searchAparLib.wss?lc=en&cc=US

You will need an IBMLink user ID to use this site.

To get access to this capability before a new version of WebSphere MQ Internet pass-thru is released, contact your support organization for an updated SSLite package. See the APAR for details.

After the installation of the updates, SSL 3.0 and TLS 1.0 protocols will interoperate between WebSphere MQ and MQIPT.

10.6 Summary

The B2B environment imposes additional requirements on top of normal WebSphere MQ practices. Many of these are illustrated in this chapter. For a specific B2B connection between two companies, all or only some of these protections might be used.

For example, you might choose to trust the firewall builders and not validate incoming IP addresses using the itsoBlockExit with WebSphere MQ Internet pass-thru.

You might be sufficiently confident of your business partner, or of the security controls within your own queue managers, that using an exit to examine and modify incoming or outgoing messages is deemed to be not needed.

However, it is important to realize that using WebSphere MQ messaging to intercommunicate with another company does present a different risk profile than messaging within a single company or division, and that systems need to be designed and built to mitigate these additional risks.

11

Scenario: Fine-grained cluster security

In an IBM WebSphere MQ point-to-point network, each remote queue manager is associated with a different receiver channel. This allows the administrator to authorize each remote node separately by placing a unique user ID in the channel's MCAUSER attribute. However, when implementing a cluster, all of the remote queue managers use the same cluster receiver channel. The result is that the fine-grained security possible with point-to-point networks is lost when upgrading to a WebSphere MQ cluster.

It is possible to address the situation by writing a channel exit but using an exit can be challenging because the exit configuration is among the channel attributes that are propagated throughout the cluster. One approach to address this problem is to use a Channel Auto-Definition (CHAD) exit. However, using an exit of either type requires that you write or acquire them separately.

This chapter presents one approach for securing a typical WebSphere MQ cluster environment using channel authentication records.

The chapter contains the following topics:

- Scenario overview
- Authorizing access using the authority context of user IDs
- Authorizing access using queue manager name mapping
- Using SSL for mutual authentication
- Authorizing access with X.509 DN mapping
- Authorizing access with X.509 and IP address mapping
- Considerations for large clusters
- Summary

11.1 Scenario overview

This scenario shows how security can be implemented in a WebSphere MQ cluster environment. The implementation has these major aspects:

- Controlling access to the cluster environment, specifically allowing authorized queue managers to connect to one another and to deny connections for any others
- Providing sufficient privileges for the cluster and its queue managers to operate, but no more, in order to circumvent misuse of authority, either inadvertently or on purpose
- Enforcing mutual authentication between queue managers to assure each other that the partner is genuine and to mitigate impersonations and spoofing

A simple approach to security in a clustered environment would be to devise a single access control role for all members. However, a single access control role would need to have a very broad access scope to allow each and every cluster member to perform its role as well as the roles of all other members within the cluster. The end result would be that one member could potentially have full control over every member within the cluster and that is not acceptable.

With classic WebSphere MQ sender/receiver channel pairs, the administrator has the option to place a different MCAUSER value on each receiver channel definition. Because the MCAUSER value determines which authorization rules apply to a connection, this allows fine-grained security capable of differentiating between any two remote queue managers. Prior to WebSphere MQ v7.1 when these same queue managers were enrolled in a WebSphere MQ cluster, this granularity was not possible unless security exits were used. Because there was only one cluster-receiver channel definition for all remote queue managers in the cluster, only one value for MCAUSER was possible, even though there could be many channel instances active at once.

WebSphere MQ v7.1 introduced channel authentication records that provide the capability to define rules about how inbound connections into the queue managers should be treated. Inbound connections might be client channels or queue manager to queue manager channels. The rules can dictate whether connections are allowed or blocked as well as provide the ability to change the user ID that a channel should assert. Therefore, the channel authentication record provides more precise control over the access granted to connecting systems at a channel level.

One approach to provide fine-grained access control is to associate each cluster member with an account (user ID); authorization would then be given on a per user ID and queue manager basis.

11.1.1 Scenario design

To effectively demonstrate the security techniques, the environment in this scenario consists of these components:

- Two full-repository queue managers that are dedicated to maintaining the integrity of the cluster. These queue managers do not host applications or application queues.
- Three cluster member queue managers that are application-specific queue managers. For context, assume that each of these queue managers serves a particular department or business unit and, as such, some require access to others but some do not.

Figure 11-1 on page 187 depicts the cluster configuration in this scenario.

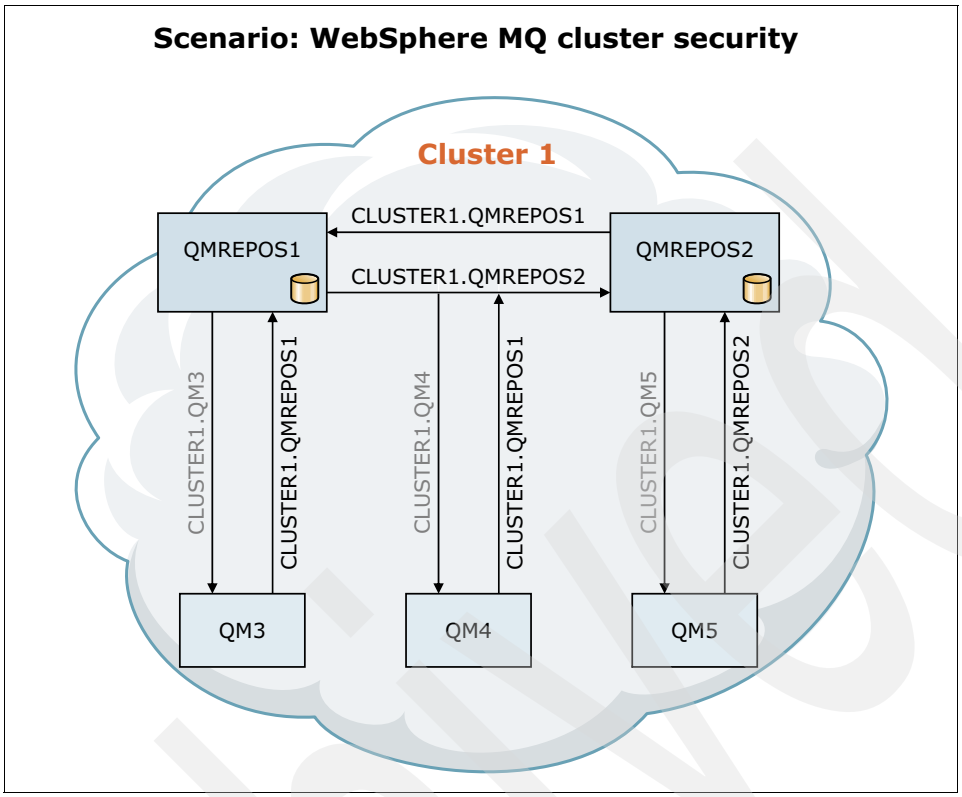


Figure 11-1 Cluster environment without security provision

This topology will be used to illustrate the following options for providing fine-grained access control to secure a WebSphere MQ cluster environment:

- Controlling access using the authority context of user IDs - This option grants access to queue managers and their objects, such as queues, on a need basis. This option is implemented by using WebSphere MQ authority profiles.
- Authorizing access by queue manager name mapping - This approach maps a remote queue manager's name to a predetermined controlled user ID. In so doing, the local queue manager dictates the level of authorization that a remote queue manager should have. This option is implemented by using WebSphere MQ channel authentication records.
- Authenticating mutually by using Secure Sockets layer (SSL) for channels - This ensures that only legitimate cluster queue managers that possess valid certificates are able to participate in the cluster. Two queue managers would need to authenticate with one another and only when both authentications are successfully carried out can the communication link between them be established.
- Authorizing access by means of mapping X.509 distinguished names (DNs) - To provide the ability to map remote queue managers' X.509 DNs to MCAUSER values at run time and on a per-instance basis. This approach ensures that only authenticated, approved queue managers are allowed to connect to one another. The implementation is based on WebSphere MQ channel authentication records.
- Authorizing access by means of mapping an X.509 DN according to an IP address - This technique is used to prevent the threats of a queue manager's name spoofing. Specifically, this technique dictates that a remote queue manager and its associated SSL certificate must connect from a specified IP address.

- Using fine-grained cluster access control. This prevents a non-full-repository queue manager from placing cluster commands onto another non-full-repository queue manager. This approach ensures that all cluster-related settings and configurations are carried out by the cluster full-repository queue managers only, as opposed to having a cluster member changing the settings of another cluster member.

This scenario demonstrates security considerations starting with an unsecured configuration. An alternative and preferred approach is a security solution that starts out with total lockdown and progressively configures the authorization profiles to arrive at the final desired settings. Adding security to an unsecured cluster was chosen because this is a common requirement that has been repeatedly experienced by the authors.

11.1.2 Preparing for the scenario

To accomplish the objectives, this scenario requires these components:

- User IDs - The scenario leverages these user IDs to provide unique authorization profiles.
- Security events monitoring tool - This tool is optional but having an ability to view authority events is certainly helpful.

A number of files will be created for each queue manager in this scenario. It is a good idea to designate a directory that contains a sub-directory for each queue manager to store these files. For example, the `QM3` sub-directory would hold the definition and certificate files that are associated with queue manager `QM3`.

Scripts are used in this scenario so that the configuration can be re-created repeatedly without typographical errors.

Configuring the user IDs

User IDs `qmrepos1`, `qmrepos2`, `qm3`, `qm4`, and `qm5` are defined on all the servers in which the queue managers reside; there is a separate security role for each queue manager. For added security, it is a good idea to create these user IDs without the login privilege. For example, in Microsoft Windows, such an ID is known as a *Windows service account*.

For the purpose of testing, several test accounts (`qmtest`, `qm3test`, `qm4test`, and `qm5test`) are optionally created. These accounts are used to run several tools to validate the settings; Table 11-1 on page 189 shows the user IDs and their attributes.

Table 11-1 Prerequisite groups and accounts that are needed for the scenario

User ID	Belong to groups	Role	Comment
qmrepos1 qmrepos2 qmtest	repos	qmrepos1 and qmrepos2 are nologin accounts. qmtest is a login account.	Designated cluster admin, full access to cluster repositories and cluster members
qm3 qm3test	qm3	qm3 is nologin account. qm3test is a login account.	Cluster member serving Human Resources applications
qm4 qm4test	qm4	qm4 is nologin account. qm4test is a login account.	Cluster member serving applications group1
qm5 qm5test	qm5	qm5 is nologin account. qm5test is a login account.	Cluster member serving application group 2

On UNIX, the following commands can be used to create the required groups and user IDs.

Example 11-1 Commands to create groups and accounts

```
# groupadd repos
# useradd -g repos -s /bin/nologin qmrepos1
# useradd -g repos -s /bin/nologin qmrepos2
# useradd -g repos -s /bin/ksh qmtest
# groupadd qm3
# useradd -g qm3 -s /bin/nologin qm3
# useradd -g qm3 -s /bin/ksh qm3test
# groupadd qm4
# useradd -g qm4 -s /bin/nologin qm4
# useradd -g qm4 -s /bin/ksh qm4test
# groupadd qm5
# useradd -g qm5 -s /bin/nologin qm5
# useradd -g qm5 -s /bin/ksh qm5test
```

Monitoring security events

In order to support monitoring and problem determination during this scenario, the SupportPac MS0P is used. The SupportPac is a set of plug-ins for the WebSphere MQ Explorer that extends the capabilities for configuration and information display. Specifically, the scenario uses the event message formatting capability to gain more detailed information about security events.

As an example, an Authorization event emitted as a result of queue manager QM4 attempting to connect to QM3 is shown in Figure 11-2 on page 190.

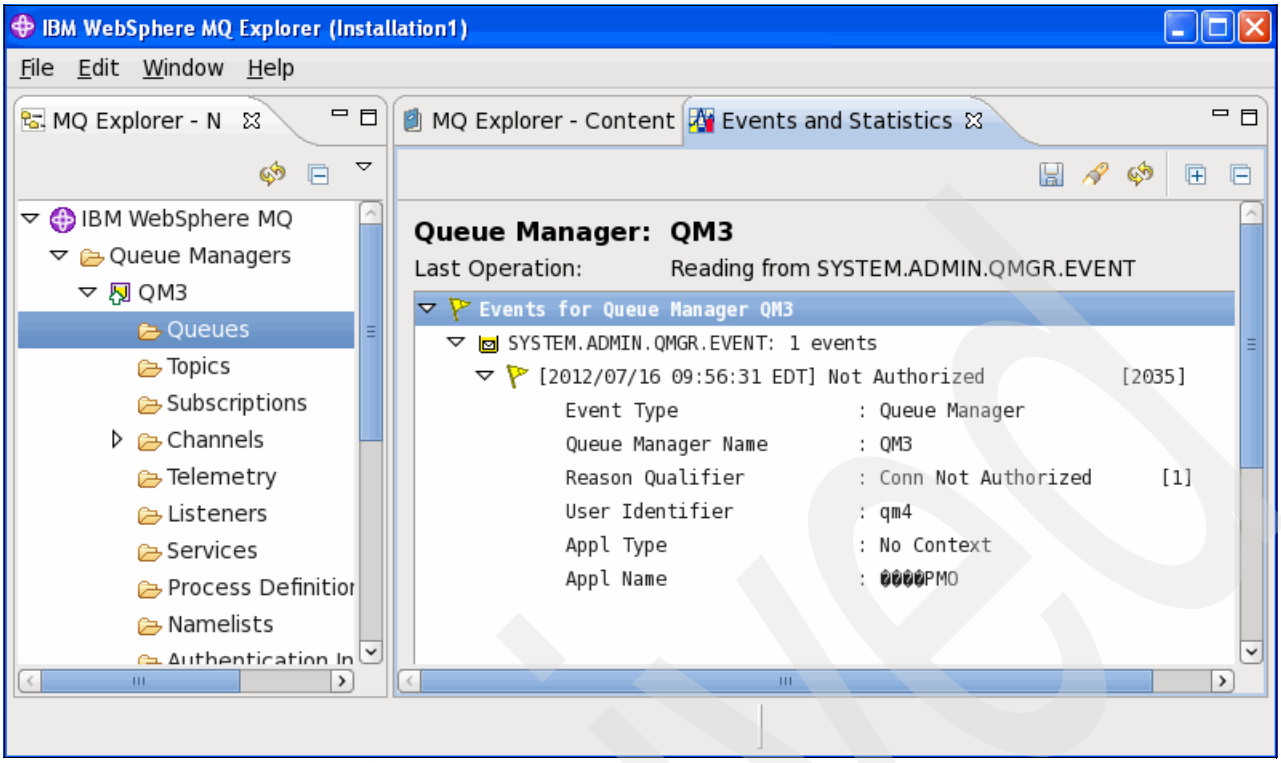


Figure 11-2 Authorization event

The event that is shown in Figure 11-2 indicates that the MCAUSER user ID of `qm4` mapped to queue manager `QM4` is not authorized to connect to queue manager `QM3`. The SupportPac MS0P and its associated documentation can be obtained from this website:

http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24011617&loc=en_US&cs=utf-8&lang=en

11.1.3 Creating the cluster

This section describes the steps that are needed to create the base topology, which consists of two full-repository queue managers and three cluster member queue managers. These queue managers represent a cluster environment to which we want to apply various security measures that are discussed in subsequent sections.

Creating two full-repository queue managers

Using two dedicated full-repository queue managers enhances cluster resiliency and robustness. In this section, two full-repository queue managers are created and named `QMREPOS1` and `QMREPOS2`. The commands are issued while logged in as the `mqm` user ID. Follow these steps:

1. Create the queue managers `QMREPOS1` and `QMREPOS2` by using the **`crtmqm`** command, for example:

   ```
   crtmqm QMREPOS1
   ```

2. Start each queue manager by using the **`strmqm`** command, for example:

   ```
   strmqm QMREPOS1
   ```

3. Configure the two queue managers as full-repository queue managers. Example 11-2 and Example 11-3 on page 191 show the **runmqsc** command with the definitions for QMREPOS1 and QMREPOS2.

Example 11-2 Full-repository cluster queue manager definitions for QMREPOS1

```
def ql(DLQ) replace
* This QM is a full-repository queue manager for cluster named CLUSTER1
alt qmgr deadq(DLQ)                            +
        repos(CLUSTER1)
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QMREPOS1) chltype(CLUSRCVR)    +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr3(1414)')
* The cluster sender channel for cluster CLUSTER1
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QMREPOS2) chltype(CLUSSDR)     +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr4(1414)')
* Allow this queue manager to listen for incoming traffic as soon as
* the queue manager is up and running
* Ensure to substitute the appropriate hostname, port number
def listener(QMREPOS1.PRIVATE)                 +
        trptype(TCP)                           +
        port(1414)                             +
        ipaddr(192.168.102.103)                +
        control(QMGR)
start listener(QMREPOS1.PRIVATE)
```

Example 11-3 Full-repository cluster queue manager definitions for QMREPOS2

```
def ql(DLQ) replace
* This QM is a full-repository queue manager for cluster named CLUSTER1
alt qmgr deadq(DLQ)                            +
        repos(CLUSTER1)
* The cluster receiver channel for cluster CLUSTER1
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QMREPOS2) chltype(CLUSRCVR)    +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr4(1414)')
* The cluster sender channel for cluster CLUSTER1
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QMREPOS1) chltype(CLUSSDR)     +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr3(1414)')
* Allow this queue manager to listen for incoming traffic as soon as
* the queue manager is up and running
* Ensure to substitute the appropriate hostname, port number
```

```
def listener(QMREPOS2.PRIVATE)
trptype(TCP)                            +
        port(1414)                              +
        ipaddr(192.168.102.104)                 +
        control(QMGR)
start listener(QMREPOS2.PRIVATE)
```

Creating three cluster member queue managers

Next, create three cluster member queue managers with the names QM3, QM4, and QM5. The commands are issued while logged in as the mqm user ID. Follow these steps:

1. Create the queue managers QM3, QM4, and QM5 by using the **crtmqm** command, for example:

   ```
   crtmqm QM3
   ```

2. Start the queue managers by using the **strmqm** command, for example:

   ```
   strmqm QM3
   ```

3. Configure the three queue managers as cluster member queue managers by using the **runmqsc** command with the definitions that are provided in Example 11-4, Example 11-5, and Example 11-6 on page 193.

   Example 11-4 shows the queue manager definition for QM3.

Example 11-4 Cluster member queue manager definition for QM3

```
define ql(DLQ) replace
alt qmgr deadq(DLQ)
* The cluster receiver channel
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QM3) chltype(CLUSRCVR)    +
        replace                         +
        cluster(CLUSTER1)               +
        conname('wmqsvr5(1414)')
* The cluster sender channel for cluster CLUSTER1
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QMREPOS1) chltype(CLUSSDR)     +
        replace                         +
        cluster(CLUSTER1)               +
        conname('wmqsvr3(1414)')
* Allow this queue manager to listen for incoming traffic as soon as
* the queue manager is up and running
* Ensure to substitute the appropriate hostname, port number
def listener(QM3.PRIVATE)               +
        trptype(TCP)                    +
        port(1414)                      +
        ipaddr(192.168.102.105)         +
        control(QMGR)
start listener(QM3.PRIVATE)
```

Example 11-5 shows the configuration for QM4.

Example 11-5 Cluster member queue manager definitions for QM4

```
def ql(DLQ) replace
alt qmgr deadq(DLQ)
* The cluster receiver channel
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QM4) chltype(CLUSRCVR)    +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr5(1415)')
* The cluster sender channel for cluster CLUSTER1
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QMREPOS2) chltype(CLUSSDR)     +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr4(1414)')
* Allow this queue manager to listen for incoming traffic as soon as
* the queue manager is up and running
* Ensure to substitute the appropriate hostname, port number
*
def listener(QM4.PRIVATE)                  +
        trptype(TCP)                           +
        port(1415)                             +
        ipaddr(192.168.102.105)                +
        control(QMGR)
start listener(QM4.PRIVATE)
```

Example 11-6 shows the queue manager definition for QM5.

Example 11-6 Cluster member queue manager definitions for QM5

```
def ql(DLQ) replace
* This QM is a cluster member queue manager in cluster named CLUSTER1
alt qmgr deadq(DLQ)
* The cluster receiver channel
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QM5) chltype(CLUSRCVR)    +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr5(1416)')
* The cluster sender channel for cluster CLUSTER1
* Ensure to substitute the appropriate hostname, port number for the
* conname option
def chl(CLUSTER1.QMREPOS2) chltype(CLUSSDR)     +
        replace                                +
        cluster(CLUSTER1)                      +
        conname('wmqsvr4(1414)')
* Allow this queue manager to listen for incoming traffic as soon as
* the queue manager is up and running
* Ensure to substitute the appropriate hostname, port number
def listener(QM5.PRIVATE)                  +
        trptype(TCP)                           +
```

```
          port(1416)                              +
          ipaddr(192.168.102.105)                 +
          control(QMGR)
   start listener(QM5.PRIVATE)
```

Initially, the QM3 queue manager has an explicit connection to the full-repository queue manager QMREPOS1, and the QM4 and QM5 queue managers initially connect to the full-repository queue manager QMREPOS2. As part of WebSphere MQ cluster capability, all cluster member queue managers would eventually be able to connect to other cluster members, including full-repository queue managers, dynamically.

At this stage, you should be able to see that all the cluster queue managers are connected to one another (Example 11-7).

Example 11-7 Cluster queue managers' status

```
$ runmqsc QMREPOS1
5724-H72 (C) Copyright IBM Corp. 1994, 2011.  ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMREPOS1.


DIS CLUSQMGR(*)
     1 : DIS CLUSQMGR(*)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(QM3)                           CHANNEL(CLUSTER1.QM3)
   CLUSTER(CLUSTER1)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(QM4)                           CHANNEL(CLUSTER1.QM4)
   CLUSTER(CLUSTER1)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(QM5)                           CHANNEL(CLUSTER1.QM5)
   CLUSTER(CLUSTER1)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(QMREPOS1)                      CHANNEL(CLUSTER1.QMREPOS1)
   CLUSTER(CLUSTER1)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(QMREPOS2)                      CHANNEL(CLUSTER1.QMREPOS2)
   CLUSTER(CLUSTER1)
```

11.2 Authorizing access using the authority context of user IDs

A default queue manager can readily accept a connection from a remote queue manager because, by default, a receiver channel has a blank MCAUSER field. The Message Channel Agent would use its default user ID. Except for the z/OS platform, for channels using TCP/IP, the default user ID is the ID that starts the listener and that typically is the ID that has WebSphere MQ full administrative access. In practice, the user ID of the listener is mqm on distributed platforms, such as UNIX. Clearly, the default situation is unacceptable and it must be prevented.

For more information about how WebSphere MQ handles blank MCAUSER, see this website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/ic11820_.htm

A common approach is to set the MCAUSER field of all inbound/receiver channels to a non-existent/invalid user ID. In this case, WebSphere MQ ignores any values that are set in the MCAUSER field that is provided by the remote queue managers and uses the hardcoded non-existent ID. The effect is that all incoming connections are denied.

To facilitate legitimate inbound connections, it is therefore necessary to create security roles that explicitly allow connections from other legitimate queue managers in the environment. These identities are then assigned to the incoming channel instances using CHLAUTH records.

In addition to restricting connections to only legitimate queue managers, it is also necessary to ensure that a remote queue manager only has enough privilege to perform its role and not more. Follow these general guidelines:

- The full-repository queue managers should have connect authority to all queue managers within the cluster.
- The full-repository queue managers are the only queue managers that need put access to the `SYSTEM.CLUSTER.COMMAND.QUEUE` of the application queue managers within a cluster.
- All cluster queue managers should have `put` access to the dead-letter queue (DLQ) of the remote queue managers if they have `connect` access to the queue manager itself. This is to ensure that any undelivered messages arriving at the remote queue managers can be put into their respective DLQs. However, it would not be appropriate to have a remote queue manager having `get` access to the local system DLQ.
- The cluster member queue managers only require access to a specific subset of application-specific queues.
- Cluster member queue managers require `put` access to the `SYSTEM.CLUSTER.COMMAND.QUEUE` on the full repository queue managers.

This scenario has these requirements:

- To allow all cluster members to connect to one another
- To differentiate the authorization roles of full-repository queue managers and cluster member queue managers because the full-repository queue managers need privileges to manage and maintain the overall cluster configuration

The subsequent subsections cover the setup in detail.

11.2.1 Creating authorization profiles for the full-repository queue managers

To set the authorization profiles for the `QMREPOS1` and `QMREPOS2` full-repository queue managers, run the **runmqsc** scripts as shown in Example 11-8 for the `QMREPOS1` and `QMREPOS2` queue managers.

Example 11-8 Set authorization profile for the full-repository queue managers

```
* Explicitly allow only the listed users to connect
set authrec objtype(qmgr) group('repos') authrmv(ALL)
set authrec objtype(qmgr) group('repos') authadd(connect,inq,setall)
set authrec objtype(qmgr) group('qm3') authrmv(ALL)
set authrec objtype(qmgr) group('qm3') authadd(connect,inq,setall)
set authrec objtype(qmgr) group('qm4') authrmv(ALL)
set authrec objtype(qmgr) group('qm4') authadd(connect,inq,setall)
set authrec objtype(qmgr) group('qm5') authrmv(ALL)
set authrec objtype(qmgr) group('qm5') authadd(connect,inq,setall)
```

```
* Allow only the listed users to put messages into SYSTEM.CLUSTER.COMMAND.QUEUE
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('repos')
authrmv(ALL)
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('repos')
authadd(put,setall)
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('qm3')
authrmv(ALL)
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('qm3')
authadd(put,setall)
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('qm4')
authrmv(ALL)
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('qm4')
authadd(put,setall)
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('qm5')
authrmv(ALL)
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('qm5')
authadd(put,setall)

* Allow listed users to put messages to the system DLQ only, i.e. no read/browse
set authrec profile(DLQ) objtype(queue) group('repos') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('repos') authadd(put,setall)
set authrec profile(DLQ) objtype(queue) group('qm3') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('qm3') authadd(put,setall)
set authrec profile(DLQ) objtype(queue) group('qm4') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('qm4') authadd(put,setall)
set authrec profile(DLQ) objtype(queue) group('qm5') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('qm5') authadd(put,setall)
```

The commands have these effects:

- The queue managers (QMREPOS1 and QMREPOS2) can be connected if the user ID belongs to group repos, qm3, qm4, or qm5. This allows cluster operations to be performed on all queue managers belonging to the clusters, for example, the full-repository queue manager QMREPOS2 is able to connect to the QMREPOS1 via its cluster-receiver channel CLUSTER1.QMREPOS1.
- A user ID that belongs to groups repos, qm3, qm4, and qm5 can put messages into the SYSTEM.CLUSTER.COMMAND.QUEUE. This allows cluster member queue managers to communicate with the cluster full-repository queue managers QMREPOS1 and QMREPOS2.

Tip: Failure to allow authorized user IDs to put messages into this queue would have these results:

- Unauthorized events are being generated in the SYSTEM.ADMIN.QMGR.QUEUE, if the Authority Events feature is enabled. These events can be monitored if you have configured the SupportPac MS0P plug-ins.
- Cluster-related command messages are being put into the DLQ at the destination queue manager. The dead-letter header MQDLH would contain a reason code of 2035 (0x7F3) MQRC_NOT_AUTHORIZED.
- Cluster operations are failing to work properly as the result of the SYSTEM.CLUSTER.COMMAND.QUEUE not being accessible to the cluster members.

- Aside from explicitly allowing authorized used IDs to have limited access to the `SYSTEM.CLUSTER.COMMAND.QUEUE`, no other SYSTEM queues are accessible even to the authorized user IDs. The restriction prevents rogue applications (impersonating a user ID belonging to the `repos` group, for example) to have full control of the queue managers if the `SYSTEM.ADMIN.COMMAND.QUEUE` is accessible to them.
- With these queue managers being full-repository queue managers, we also want to restrict access to their DLQs to any users belonging to the `repos`, `qm3`, `qm4`, and `qm5` groups so that messages can be put into them (for any undeliverable messages caused by a cluster operation). At the same time, a user ID belonging to these groups cannot perform message browsing or deletions, which are typically carried by a designated user ID or a user ID with WebSphere MQ administrator privilege, for example, `mqm`.
- The `inq` authority for the QMGR object is required to allow an authorized remote system to discover the local queue manager's properties, such as the name of the DLQ.

> **Tip:** Failure to set `inq` authority for the queue manager object would result in the following error:
>
> ```
> AMQ8077: Entity 'qmrepos2    ' has insufficient authority to access object
> 'QMREPOS1'.
>
> EXPLANATION:
> The specified entity is not authorized to access the required object. The
> following requested permissions are unauthorized: inq
>
> AMQ9509: Program cannot open queue manager object.
>
> EXPLANATION:
> The attempt to open either the queue or queue manager object 'QMREPOS1' on
> queue manager 'QMREPOS1' failed with reason code 2035.
> ```

Instead of using the **runmqsc** scripts, you can also use the equivalent WebSphere MQ control command **setmqaut**, for example:

```
setmqaut -m QMREPOS1 -g repos -t qmgr -all +connect +setall
```

This command is the equivalent of:

```
set authrec objtype(qmgr) group('repos') authadd(connect,setall)
```

> **Tip:** Authorization settings are cumulative. It is therefore a good practice to reset the authorization settings for a profile to remove any prior settings before setting new ones.

11.2.2 Creating authorization profiles for the cluster member queue managers

The next step is to set the authorization profile for the queue managers `QM3`, `QM4`, and `QM5`.

The **runmqsc** commands for QM3 are shown in Example 11-9.

Example 11-9 Set authorization profiles for queue manager QM3

```
* Explicitly allow only the listed groups to connect
* Note: qm4 group is not allowed to connect to this queue manager
set authrec objtype(qmgr) group('repos') authrmv(ALL)
set authrec objtype(qmgr) group('repos') authadd(connect,inq,setall)
```

```
set authrec objtype(qmgr) group('qm5') authrmv(ALL)
set authrec objtype(qmgr) group('qm5') authadd(connect,inq,setall)

* Allow only the cluster admins to put messages into SYSTEM.CLUSTER.COMMAND.QUEUE

* Note: qm4 and qm5 groups are not allowed to put messages into SYSTEM queues
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('repos')
authadd(put,setall)

* Allow listed users to put messages to the system DLQ only, i.e. no read/browse
set authrec profile(DLQ) objtype(queue) group('repos') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('repos') authadd(put,setall)
set authrec profile(DLQ) objtype(queue) group('qm5') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('qm5') authadd(put,setall)
```

In summary, the authorization profile for QM3 would have these characteristics:

- Allow members of the `repos` group to connect to this queue manager so that the two full-repository queue managers can manage `QM3`.
- Allow members of the `qm5` group to connect to `QM3` so that an application connecting to `QM5` can put messages to an application-specific cluster queue residing on `QM3`.
- Allow only the `repos` group to have `put` and `setall` authorities for the `SYSTEM.CLUSTER.COMMAND.QUEUE`. Note that `qm3`, `qm4`, and `qm5` groups are not granted access to this queue, because they do not need access.
- Allow only members of the `repos` and `qm5` groups to have `put` and `setall` authorities for the system DLQ. `QM4` does not need to access this queue manager; therefore, there are no authority records associated with `qm4`. An attempt to connect from `QM4` to this queue manager would be denied.

The **runmqsc** commands for `QM4` are shown in Example 11-10.

Example 11-10 Set authorization profiles for queue manager QM4 using runmqsc commands

```
* Explicitly allow only the listed groups to connect
set authrec objtype(qmgr) group('repos') authrmv(ALL)
set authrec objtype(qmgr) group('repos') authadd(connect,inq,setall)
set authrec objtype(qmgr) group('qm5') authrmv(ALL)
set authrec objtype(qmgr) group('qm5') authadd(connect,inq,setall)

* Only allow cluster admin to access the cluster command queue.
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('repos')
authadd(put,setall)

set authrec profile(SYSTEM.ADMIN.COMMAND.QUEUE) objtype(queue) group('repos')
authrmv(all)

* Allow listed users to put messages to the system DLQ only, i.e. no read/browse
set authrec profile(DLQ) objtype(queue) group('repos') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('repos') authadd(put,setall)
set authrec profile(DLQ) objtype(queue) group('qm5') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('qm5') authadd(put,setall)
```

The **runmqsc** commands for QM5 are shown in Example 11-11.

Example 11-11 Set authorization profiles for queue manager QM5 using runmqsc commands

```
set authrec objtype(qmgr) group('repos') authrmv(ALL)
set authrec objtype(qmgr) group('repos') authadd(connect,inq,setall)

set authrec objtype(qmgr) group('qm3') authrmv(ALL)
set authrec objtype(qmgr) group('qm3') authadd(connect,inq,setall)
set authrec objtype(qmgr) group('qm4') authrmv(ALL)
set authrec objtype(qmgr) group('qm4') authadd(connect,inq,setall)

* Only allow cluster admin to access the cluster command queue.
set authrec profile(SYSTEM.CLUSTER.COMMAND.QUEUE) objtype(queue) group('repos')
authadd(put,setall)

* Allow listed users to put messages to the system DLQ only, i.e. no read/browse
set authrec profile(DLQ) objtype(queue) group('qm3') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('qm3') authadd(put,setall)
set authrec profile(DLQ) objtype(queue) group('qm4') authrmv(ALL)
set authrec profile(DLQ) objtype(queue) group('qm4') authadd(put,setall)
```

11.2.3 Setting the MCAUSER user to block unauthorized access

The basic cluster environment built so far would allow a remote queue manager to connect to it because the MCAUSER attribute of the cluster receiver channels is blank. The (local) Message Channel Agent would use its default user ID. For channels using TCP/IP on platforms other than z/OS, the default user ID is the ID that starts the listener and typically that ID has WebSphere MQ full administrative access. The end result means that a remote queue manager has full administrative authority over the local queue manager. Clearly, this situation is unacceptable.

> **Important:** As of WebSphere MQ v7.1, connections using a Server Connection Channel SVRCONN are blocked by default.

Prior to performing further authorization configuration, it is considered to be a good practice to set the MCAUSER of all the receiver channels to an invalid user ID with the effect of blocking all incoming connections by default. Any legitimate connections would then need to be explicitly allowed by means of setting the CHLAUTH rules.

To change the definitions of all cluster receiver channels for all cluster queue managers to have an invalid user ID of *NOACCESS specified for the MCAUSER attribute, issue the following **runmqsc** commands:

- For QMREPOS1:

  ```
  alt chl(CLUSTER1.QMREPOS1) chltype(CLUSRCVR) mcauser(‘*NOACCESS’)
  ```

- For QMREPOS2:

  ```
  alt chl(CLUSTER1.QMREPOS2) chltype(CLUSRCVR) mcauser(‘*NOACCESS’)
  ```

- For QM3:

  ```
  alt chl(CLUSTER1.QM3) chltype(CLUSRCVR) mcauser(‘*NOACCESS’)
  ```

- For QM4:

  ```
  alt chl(CLUSTER1.QM4) chltype(CLUSRCVR) mcauser('*NOACCESS')
  ```

- For QM5:

  ```
  alt chl(CLUSTER1.QM5) chltype(CLUSRCVR) mcauser('*NOACCESS')
  ```

11.2.4 Validating the authorization settings

To confirm the settings, use the **runmqsc display AUTHREC** commands as shown in Example 11-12 on page 200.

Example 11-12 Use the runmqsc DISPLAY command to list authority records

```
dis authrec profile(DLQ)
     1 : dis authrec profile(DLQ)
AMQ8864: Display authority record details.
   PROFILE(DLQ)                            ENTITY(mqm)
   ENTTYPE(GROUP)                          OBJTYPE(QUEUE)
   AUTHLIST(BROWSE,CHG,CLR,DLT,DSP,GET,INQ,PUT,PASSALL,PASSID,SET,SETALL,SETID)
AMQ8864: Display authority record details.
   PROFILE(DLQ)                            ENTITY(repos)
   ENTTYPE(GROUP)                          OBJTYPE(QUEUE)
   AUTHLIST(PUT,SETALL)
AMQ8864: Display authority record details.
   PROFILE(DLQ)                            ENTITY(qm3)
   ENTTYPE(GROUP)                          OBJTYPE(QUEUE)
   AUTHLIST(PUT,SETALL)
AMQ8864: Display authority record details.
   PROFILE(DLQ)                            ENTITY(qm4)
   ENTTYPE(GROUP)                          OBJTYPE(QUEUE)
   AUTHLIST(PUT,SETALL)
AMQ8864: Display authority record details.
   PROFILE(DLQ)                            ENTITY(qm5)
   ENTTYPE(GROUP)                          OBJTYPE(QUEUE)
   AUTHLIST(PUT,SETALL)
```

You can see from this output that the following conditions are true:

- The group mqm has full authority on the dead-letter queue DLQ. This is the default authority.
- The settings have been made for group repos to only have the authority to put message and set all context for the dead-letter queue DLQ.

To confirm that the authorization profiles have taken effect, follow these steps:

1. Log on to a server where a queue manager (QMREPOS1, for example) resides, with a test user ID, for example, qmtest.
2. Put a test message into the dead-letter queue DLQ using the supplied sample program amqsput, and the operation should be successful, for example:

   ```
   /opt/mqm/samp/bin/amqsput DLQ QMREPOS1
   ```

3. Confirm that attempts to get test messages from the dead-letter queue DLQ using the supplied sample program amqsget would fail, for example:

   ```
   /opt/mqm/samp/bin/amqsget DLQ QMREPOS1
   ```

4. Confirm that attempts to put a test message into the `SYSTEM.ADMIN.COMMAND.QUEUE` using the supplied sample program amqsput would fail, for example:

   ```
   /opt/mqm/samp/bin/amqsput SYSTEM.ADMIN.COMMAND.QUEUE QMREPOS1
   ```

At this stage, there should be authorization errors logged in all the receiving queue managers' error logs, such as those errors shown in Example 11-13.

Example 11-13 Incoming connection denied due to unauthorized MCAUSER user ID

```
AMQ5653: The user '*NOACCESS' is not defined.

AMQ9557: Queue Manager User ID initialization failed.

AMQ9999: Channel 'CLUSTER1.QMREPOS1' to host 'wmqsvr5 (192.168.102.105)' ended
abnormally.
```

Errors are also logged on the sending queue managers indicating that messages were not sent correctly, as shown in Example 11-14.

Example 11-14 Error at the sending end due to unauthorized MCAUSER user ID

```
AMQ9506: Message receipt confirmation failed.
```

The errors shown in Example 11-13 on page 201 and Example 11-14 are expected and intentional, which means that all the queue managers are now configured to deny incoming connections, by default. The subsequent sections describe techniques to selectively allow incoming connections.

11.2.5 Summary

We have restricted the access of authorized user IDs so that each would only have sufficient authority to perform its required role but no more than that. For example, a cluster member does not have cluster administration privilege nor does it have the capability to issue WebSphere MQ administrative commands to another cluster member.

We have also blocked all incoming connections by means of setting the MCAUSER user ID of receiver channels to an invalid user ID.

These conditions are true, in general:

- Full-repository queue managers would typically only require limited access to these queues:
  - `SYSTEM.CLUSTER.COMMAND.QUEUE`
  - Dead-letter queue
- Cluster member queue managers do not normally need access to these queues:
  - `SYSTEM.CLUSTER.TRANSMIT.QUEUE`
  - `SYSTEM.CLUSTER.COMMAND.QUEUE` (except on the full-repository queue managers)

Tip: In WebSphere MQ v7.1 or later, a new attribute `ClusterQueueAccessControl` in the `Security` stanza of `qm.ini` has been introduced to specify the method of access control for cluster queues and fully qualified queues hosted on cluster queue managers. The stanza accepts one of these values:

- `RQMName` - The profiles checked for access control of remotely hosted queues are named *queues* or named *queue manager profiles*. WebSphere MQ would use custom security profiles for authorization access of remote hosted queues.
- `Xmitq` - Profiles are checked for access control of remotely hosted queues that are resolved to the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

The default if not specified is `Xmitq` and has the same behavior as previous WebSphere MQ versions. In practice, `RQMName` provides more granular control of access and should always be used in preference to `Xmitq`, providing that the option is available.

For further details, see this website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fa12555_.htm

Similar capability could be achieved in previous versions by creating QAlias definitions and granting access to them. The target of the alias could have the same name as the alias itself when resolution was to a cluster queue.

11.3 Authorizing access using queue manager name mapping

To prevent a remote queue manager from exploiting the situation where the MCAUSER attribute is left blank to assert administrative authority, this section configures the cluster queue managers so that each authorized queue manager is mapped to a specific authorized user ID, which has its privileges according to its role configured as described in 11.2, “Authorizing access using the authority context of user IDs” on page 194.

Prior to proceeding with the setup, ensure that the following conditions are true:

- The authorization profiles have been created and configured as described in 11.2, “Authorizing access using the authority context of user IDs” on page 194.
- You are blocking connections to queue managers by default as described in “Setting the MCAUSER user to block unauthorized access” on page 199.

The mapping of a queue manager to a user ID is accomplished by using CHLAUTH records.

11.3.1 Creating CHLAUTH records for the full-repository queue managers

Follow these steps:

1. Create CHLAUTH records for the full-repository queue managers `QMREPOS1` and `QMREPOS2`.
2. Create authorization profiles for the full-repository queue managers `QMREPOS1` and `QMREPOS2`.

3. On the QMREPOS1 queue manager, define new CHLAUTH records by using the **runmqsc** scripts that are shown in Example 11-15.

Example 11-15 runmqsc scripts to create CHLAUTH record for QMREPOS1 and QMREPOS2

```
alt qmgr chlauth(enabled)
set chlauth(CLUSTER1.QMREPOS1) type(qmgrmap) qmname('QMREPOS1')
mcauser('qmrepos1')
set chlauth(CLUSTER1.QMREPOS1) type(qmgrmap) qmname('QMREPOS2')
mcauser('qmrepos2')
set chlauth(CLUSTER1.QMREPOS1) type(qmgrmap) qmname('QM3') mcauser('qm3')
set chlauth(CLUSTER1.QMREPOS1) type(qmgrmap) qmname('QM4') mcauser('qm4')
set chlauth(CLUSTER1.QMREPOS1) type(qmgrmap) qmname('QM5') mcauser('qm5')
```

The following explanations refer to Example 11-15 on page 203:

– The first command ensures that the channel authentication feature for the queue manager is enabled.

– The second command creates a new CHLAUTH authentication record for channel CLUSTER1.QMREPOS1 that maps a connection from queue manager QMREPOS1 by this channel to an MCAUSER user ID of qmrepos1.

– The third command creates a new CHLAUTH authentication record for channel CLUSTER1.QMREPOS1 that maps a connection from queue manager QMREPOS2 to an MCAUSER user ID of qmrepos2.

– The next three commands create new CHLAUTH records for each incoming connection from QM3, QM4, and QM5. Unlike the role of QMREPOS1 or QMREPOS2, these cluster members should not be given privilege to manage cluster membership.

The **chlauth** command takes a generic channel name, which can be fully spelled out as in this case or it can take a pattern. A pattern can include one or more asterisks (*) as a wildcard that matches the channel name.

4. Use the steps that are described for QMREPOS1 to configure QMREPOS2 using the **runmqsc** scripts that are shown in Example 11-15 on page 203.

5. To ensure that the changes introduced here take effect immediately, restart all cluster-sender channels. For example, to restart the cluster-sender CLUSTER1.QMREPOS1 from the QMREPOS2 queue manager, use the **runmqsc** commands:

```
stop chl(CLUSTER1.QMREPOS1) status(inactive)
start chl(CLUSTER1.QMREPOS1)
```

6. Confirm that the cluster-receiver channels are now running under their respective MCAUSER user ID. Example 11-16 shows that the cluster-receiver CLUSTER1.QMREPOS1 on QMREPOS1, initiated by QMREPOS2, is now correctly running under the MCAUSER ID of qmrepos2.

Example 11-16 Confirming the new configuration

```
dis chs(CLUSTER1.QMREPOS1) mcauser
     5 : dis chs(CLUSTER1.QMREPOS1) mcauser
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QMREPOS1)              CHLTYPE(CLUSRCVR)
   CONNAME(192.168.102.104)                CURRENT
   MCAUSER(qmrepos2)                       RQMNAME(QMREPOS2)
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

11.3.2 Configuring the cluster member queue managers

The next step is to configure the cluster member queue managers:

1. On the QM3 queue manager, define new CHLAUTH records using the **runmqsc** scripts that are shown in Example 11-17 on page 204.

   Example 11-17 runmqsc scripts to create CHLAUTH record for QM3

   ```
   alt qmgr chlauth(enabled)
   set chlauth(CLUSTER1.QM3) type(qmgrmap) qmname('QMREPOS*') mcauser('qmrepos1')
   set chlauth(CLUSTER1.QM3) type(qmgrmap) qmname('QM5') mcauser('qm5')
   ```

   Refer to the scripts in Example 11-17:

   – The first command ensures that the channel authentication feature is enabled for the queue manager.

   – The second command creates a new CHLAUTH authentication record named CLUSTER1.QM3 that maps any channel connecting from queue managers that have the names beginning with QMREPOS to an MCAUSER qmrepos1. That is, a connection from QMREPOS1 or QMREPOS2 would be mapped to an MCAUSER of qmrepos1. This would effectively allow the two full-repository queue managers to manage this cluster member queue manager.

   – The third command creates a new CHLAUTH record for incoming connections from QM5. This mapping is intended to demonstrate that we can selectively allow one cluster member to connect to another on a per need basis. Here, we would allow QM5 to connect to QM3, but not allow QM4 to connect to QM3. Unlike the role of QMREPOS1 or QMREPOS2, QM4 and QM5 should not be given privilege to manage cluster member QM3.

2. Use the **runmqsc** scripts that are shown in Example 11-18 to create the CHLAUTH records for QM4.

   Example 11-18 runmqsc scripts to create CHLAUTH record for QM4

   ```
   alt qmgr chlauth(enabled)
   set chlauth(CLUSTER1.QM4) type(qmgrmap) qmname('QMREPOS*') mcauser('qmrepos1')
   set chlauth(CLUSTER1.QM4) type(qmgrmap) qmname('QM5') mcauser('qm5')
   ```

3. Use the **runmqsc** scripts that are shown in Example 11-19 to create the CHLAUTH records for QM5.

   Example 11-19 runmqsc scripts to create CHLAUTH record for QM5

   ```
   alt qmgr chlauth(enabled)
   set chlauth(CLUSTER1.QM5) type(qmgrmap) qmname('QMREPOS*') mcauser('qmrepos1')
   set chlauth(CLUSTER1.QM5) type(qmgrmap) qmname('QM3') mcauser('qm3')
   set chlauth(CLUSTER1.QM5) type(qmgrmap) qmname('QM4') mcauser('qm4')
   ```

   With the intention of allowing applications connecting to QM5 to access cluster queues residing on either QM3 or QM4, or both, the configuration that is shown in Example 11-19 provides authorization mappings for both queue managers so that responses can be returned to QM5.

11.3.3 Validating the authorization settings

There are several displays that you can use to confirm the authorization settings are configured:

1. To confirm the settings, use the **`runmqsc display CHLAUTH`** command as shown in Example 11-20.

Example 11-20 Display CHLAUTH record

```
dis chlauth(CLUSTER1.QMREPOS1) TYPE(QMGRMAP)
     4 : dis chlauth(CLUSTER1.QMREPOS1) TYPE(QMGRMAP)
AMQ8878: Display channel authentication record details.
   CHLAUTH(CLUSTER1.QMREPOS1)              TYPE(QMGRMAP)
   DESCR( )                                CUSTOM( )
   ADDRESS( )                              QMNAME(QM3)
   MCAUSER(qm3)                            USERSRC(MAP)
   WARN(NO)                                ALTDATE(2012-07-12)
   ALTTIME(17.15.35)
AMQ8878: Display channel authentication record details.
   CHLAUTH(CLUSTER1.QMREPOS1)              TYPE(QMGRMAP)
   DESCR( )                                CUSTOM( )
   ADDRESS( )                              QMNAME(QM4)
   MCAUSER(qm4)                            USERSRC(MAP)
   WARN(NO)                                ALTDATE(2012-07-12)
   ALTTIME(17.15.35)
AMQ8878: Display channel authentication record details.
   CHLAUTH(CLUSTER1.QMREPOS1)              TYPE(QMGRMAP)
   DESCR( )                                CUSTOM( )
   ADDRESS( )                              QMNAME(QM5)
   MCAUSER(qm5)                            USERSRC(MAP)
   WARN(NO)                                ALTDATE(2012-07-12)
   ALTTIME(17.15.35)
```

The output shows the following information:

– An incoming connection request from `QM3` using the receiver channel `CLUSTER1.QMREPOS1` would have its MCAUSER user ID mapped to `qm3`.

– An incoming connection request from `QM4` using the receiver channel `CLUSTER1.QMREPOS1` would have its MCAUSER user ID mapped to `qm4`.

– An incoming connection request from `QM5` using the receiver channel `CLUSTER1.QMREPOS1` would have its MCAUSER user ID mapped to `qm5`.

With the channel authorization records configured, all the queue managers should now be able to connect to one another according to their respective roles.

2. On each cluster member queue manager, start the cluster sender channels and confirm that their status is `RUNNING` with the commands in Example 11-21.

Example 11-21 runmqsc commands to start up cluster sender channels and display their status

```
start CHANNEL(CLUSTER1.QM3)
     8 : start CHANNEL(CLUSTER1.QM3)
AMQ8018: Start WebSphere MQ channel accepted.
START CHANNEL(CLUSTER1.QM4)
     9 : START CHANNEL(CLUSTER1.QM4)
AMQ8018: Start WebSphere MQ channel accepted.
START CHANNEL(CLUSTER1.QM5)
```

```
    10 : START CHANNEL(CLUSTER1.QM5)
AMQ8018: Start WebSphere MQ channel accepted.
START CHANNEL(CLUSTER1.QMREPOS2)
    11 : START CHANNEL(CLUSTER1.QMREPOS2)
AMQ8018: Start WebSphere MQ channel accepted.
DIS CHS(*)
    12 : DIS CHS(*)
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QM3)                   CHLTYPE(CLUSSDR)
   CONNAME(192.168.102.105(1414))          CURRENT
   RQMNAME(QM3)                            STATUS(RUNNING)
   SUBSTATE(MQGET)                         XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QM4)                   CHLTYPE(CLUSSDR)
   CONNAME(192.168.102.105(1415))          CURRENT
   RQMNAME(QM4)                            STATUS(RUNNING)
   SUBSTATE(MQGET)                         XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QM5)                   CHLTYPE(CLUSSDR)
   CONNAME(192.168.102.105(1416))          CURRENT
   RQMNAME(QM5)                            STATUS(RUNNING)
   SUBSTATE(MQGET)                         XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QMREPOS2)              CHLTYPE(CLUSSDR)
   CONNAME(192.168.102.104(1414))          CURRENT
   RQMNAME(QMREPOS2)                       STATUS(RUNNING)
   SUBSTATE(MQGET)                         XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
```

3. Confirm the status of the cluster receiver channels to ensure that they are running under the respective MCAUSER user ID. The correct status for the cluster receiver channel in `QMREPOS1` initiated by `QM3` should indicate that the channel is running under the MCAUSER user ID of `qm3` (Example 11-22).

Example 11-22 Correct MCAUSER for the receiver channel initiated by QM3

```
DIS CHS(*) MCAUSER
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QMREPOS1)              CHLTYPE(CLUSRCVR)
   CONNAME(192.168.102.105)                CURRENT
   MCAUSER(qm3)                            RQMNAME(QM3)
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

The final verification step can provide important information if the configuration is not correct.

For example, the status of the cluster receiver channel in `QMREPOS1` initiated by `QM3` shown in Example 11-23 is incorrect because it is still running under the default `mqm` user ID.

Example 11-23 Incorrect MCAUSER user ID

```
DIS CHS(*)
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QMREPOS1)              CHLTYPE(CLUSRCVR)
   CONNAME(192.168.102.105)                CURRENT
   MCAUSER(mqm)                            RQMNAME(QM3)
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

To correct the status, ensure that the CHLAUTH record for the `CLUSTER1.QMREPOS1 on QMREPOS1` is correct and restart the channel if necessary.

Another error that you may encounter is a channel that is not running. If that is the case, review the queue manager's log to determine the reason. A typical error shown in Example 11-24 would result in channels failing to connect.

Example 11-24 Failing to connect to a channel

```
AMQ8077: Entity 'qm3         ' has insufficient authority to access object
'QMREPOS1'.
```

The error in Example 11-24 on page 207 occurs on the full-repository queue manager `QMREPOS1` as a result of missing an Authority record, AUTHREC, to explicitly allow remote queue manager `QM3`, mapping to user ID `qm3`, to connect to `QMREPOS1`.

Example 11-25 shows an error that has occurred on the full-repository queue manager `QMREPOS2` as a result of a missing Authority record to explicitly allow remote queue manager `QM3`, mapping to user ID `qm3`, to have `put` and `setall` accesses.

Example 11-25 Failing to connect to a channel due to no read authorization

```
AMQ8077: Entity 'qm3         ' has insufficient authority to access object
'SYSTEM.CLUSTER.COMMAND.QUEUE'.
```

You might see an `AMQ8077` error that indicates a user ID does not have sufficient authorization to start a cluster channel, for example, a remote queue manager does not have the authorization to start the channel CLUSTER1.QMREPOS1. In this case, determine the user ID that is associated with the MCAUSER and then determine the group for the user ID. Ensure that the authorization profile exists for that group.

Tip: To be able to monitor authorization events, turn on the queue manager's Authorization Event feature:

```
alt qmgr authorev(enabled)
```

All the authorization events that are generated are in the `SYSTEM.ADMIN.QMGR.EVENT` queue and you can use the MS0P SupportPac plug-in in WebSphere MQ Explorer to monitor these events.

11.3.4 Summary

We have configured the cluster environment so that only queue managers within the cluster can connect to one another by means of mapping queue managers to their respective authorized user IDs and so that each user ID has a specific authorization profile. Additionally, we also restricted the access of each queue manager so that each queue manager would only have sufficient authority to perform its respective role but no more. For example, a cluster member does not have cluster administration privilege nor does it have the capability to issue WebSphere MQ administrative commands to another cluster member.

11.4 Using SSL for mutual authentication

This section describes the required steps to allow one queue manager to connect to another by means of mutual authentication. For the concept of using Secure Sockets Layer (SSL) in WebSphere MQ, see the link:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs00140_.htm

11.4.1 Creating certificates for the cluster queue managers

For each queue manager in the cluster, a certificate is requested using the information that is shown in Table 11-2 on page 208.

Table 11-2 SSL certificate details

QM name	Label	DN
QMREPOS1	ibmwebspheremqqmrepos1	CN=qmrepos1,OU=SA-W216 Residency,O=IBM ITSO,C=US
QMREPOS2	ibmwebspheremqqmrepos2	CN=qmrepos2,OU=SA-W216 Residency,O=IBM ITSO,C=US
QM3	ibmwebspheremqqm3	CN=qm3,OU=SA-W216 Residency,O=IBM ITSO,C=US
QM4	ibmwebspheremqqm4	CN=qm4,OU=SA-W216 Residency,O=IBM ITSO,C=US
QM5	ibmwebspheremqqm5	CN=qm5,OU=SA-W216 Residency,O=IBM ITSO,C=US

The following instructions assume that these conditions are true:

- The user ID that is used to issue the commands is the designated WebSphere MQ administrator `mqm` since access should be restricted to the key files.
- The SSL infrastructure has been set up with a Root certificate authority and an Intermediate certificate authority.

For each cluster queue manager in this scenario, perform the following steps:

1. Use the **runmqakm** command to create a random password to be used for the keystore, as shown in Example 11-26.

   Example 11-26 Create a random password for the keystore

   ```
   runmqakm -random -create -length 125 -strong | tr -d "'" | tr -d
   '\\\$\%\`\~\&\@\!\|\\[\]" ' | cut -c 2-65 > key.passwd

   chmod g-rwx,o-rwx key.passwd
   ```

2. Use the **runmqakm** command to create a keystore for the queue manager as shown in Example 11-27.

   Example 11-27 Create a keystore

   ```
   runmqakm -fips -keydb -create -db key.kdb -pw "`cat key.passwd`" -type cms
   -stash -empty
   ```

3. Add the certificates of the required certificate authorities into the keystore as shown in Example 11-28.

Example 11-28 Add CA certificates into keystore

```
runmqakm -fips -cert -add -db key.kdb -stashed -file wmqca1.arm -label "WMQCA1
MQ Root CA" -trust enable

runmqakm -fips -cert -add -db key.kdb -stashed -file wmqca2.arm -label "WMQCA2
MQ Intermediate CA" -trust enable
```

4. Create a certificate request for the queue manager and send it to a certificate authority (CA), as shown in Example 11-29.

Example 11-29 Create a certificate request

```
runmqakm -fips -certreq -create -db key.kdb -stashed -label
ibmwebspheremqqmrepos1 -dn "CN=qmrepos1,OU=SA-W216 Residency,O=IBM ITSO,C=US"
-target qmrepos1.req -sigalg sha256 -size 2048
```

The label must be prefixed with `ibmwebsphere` followed by the name of the queue manager (the command in Example 11-29 on page 209 refers to queue manager `QMREPOS1`) all in lowercase characters.

The output of the request must then be sent to the CA, which would return a signed certificate for the queue manager

5. Receive the signed certificate into the keystore as shown in Example 11-30.

Example 11-30 Receive certificate issued by the CA into keystore

```
$ runmqakm -fips -cert -receive -db key.kdb -stashed -file qmrepos1.cer
```

6. Display the certificates currently in the keystore file as shown in Example 11-31.

Example 11-31 Display certificates currently stored in a keystore

```
$ runmqakm -fips -cert -list -db key.kdb -stashed
Certificates found
* default, - personal, ! trusted
!       WMQCA1 MQ Root CA
!       WMQCA2 MQ Intermediate CA
-       ibmwebspheremqqmrepos1
```

7. Copy the final keystore and its associated files over to the default WebSphere MQ keystore sub-directory of a queue manager. For example, for queue manager `QMREPOS1`, it would be `/var/mqm/qmgrs/QMREPOS1/ssl`.

Example 11-32 Copy keystore and its associated files over to a queue manager's sub-directory

```
$ cp key.* /var/mqm/qmgrs/QMREPOS1/ssl
```

8. Repeat these steps to create certificates for the remaining queue managers.

11.4.2 Configuring the cluster queue managers to use SSL

In this section, we are going to configure the queue managers so that a connection can only be established if the exchange of SSL certificates is successful.

Update all queue managers in the cluster to use SSL. The commands are shown in Example 11-33 for QMREPOS1.

Example 11-33 runmqsc commands to enable SSL for QMREPOS1

```
alt qmgr sslfips(yes)
alt CHL(CLUSTER1.QMREPOS1) CHLTYPE(CLUSRCVR)            +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
alt CHL(CLUSTER1.QMREPOS2) CHLTYPE(CLUSSDR)             +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
refresh security type(ssl)
```

The following descriptions refer to Example 11-33:

- The first command updates the queue manager to enforce compliance to the Federal Information Processing Standards (FIPS), which mandates security standards.
- The second and third commands enable both the sender and receiver channels to use the cypherspec of TLS_RSA_WITH_AES_256_CBC_SHA. It is essential that the CipherSpec at both ends of a channel match each other.
- The last command forces the changes to take effect. The **refresh security type(ssl)** command can take some time to complete. It is important to wait for the following response before proceeding to ensure that the changes have propagated across the cluster topology:

  ```
  AMQ8560: WebSphere MQ security cache refreshed.
  ```

Example 11-34 shows the commands to enable SSL for QMREPOS2.

Example 11-34 runmqsc commands to enable SSL for QMREPOS2

```
alt qmgr sslfips(yes)
alt CHL(CLUSTER1.QMREPOS2) CHLTYPE(CLUSRCVR)            +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
alt CHL(CLUSTER1.QMREPOS1) CHLTYPE(CLUSSDR)             +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
refresh security type(ssl)
```

Example 11-35 shows the commands to enable SSL for QM3.

Example 11-35 runmqsc commands to enable SSL for QM3

```
alt qmgr sslfips(yes)
alt CHL(CLUSTER1.QM3) CHLTYPE(CLUSRCVR)            +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
alt CHL(CLUSTER1.QMREPOS1) CHLTYPE(CLUSSDR)             +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
refresh security type(ssl)
```

Example 11-36 shows the commands to enable SSL for QM4.

Example 11-36 runmqsc commands to enable SSL for QM4

```
alt qmgr sslfips(yes)
alt CHL(CLUSTER1.QM4) CHLTYPE(CLUSRCVR)            +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
alt CHL(CLUSTER1.QMREPOS2) CHLTYPE(CLUSSDR)             +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
refresh security type(ssl)
```

Example 11-37 shows the commands to enable SSL for QM5.

Example 11-37 runmqsc commands to enable SSL for QM5

```
alt qmgr sslfips(yes)
alt CHL(CLUSTER1.QM5) CHLTYPE(CLUSRCVR)            +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
alt CHL(CLUSTER1.QMREPOS2) CHLTYPE(CLUSSDR)             +
        SSLCIPH(TLS_RSA_WITH_AES_256_CBC_SHA)
refresh security type(ssl)
```

11.4.3 Validating the settings

Start all the cluster sender channels of the cluster queue managers and confirm that their status is RUNNING.

To confirm that the SSL settings have taken effect, display the channel status using the **SSLCERTI** option. A channel is using SSL if you see the DN values that are associated with that channel. See Example 11-38 for an example.

Example 11-38 A running channel with SSL enabled

```
DIS CHS(*) SSLCERTI
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QMREPOS2)            CHLTYPE(CLUSSDR)
   CONNAME(192.168.102.104(1414))        CURRENT
   RQMNAME(QMREPOS2)
   SSLCERTI(CN=WMQCA2 MQ Intermediate CA,OU=SA-W216 Residency,O=IBM ITSO,C=US)
   STATUS(RUNNING)                       SUBSTATE(MQGET)
   XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
```

Having set up and configured SSL, all the queue managers should now be able to connect to one another securely by means of mutual authentication, that is, the ability to confirm the partner is genuine.

11.4.4 Summary

We have configured the cluster environment so that only authorized queue managers that possess a valid SSL certificate within the cluster can connect to one another by means of SSL mutual authentication.

11.5 Authorizing access with X.509 DN mapping

This section describes the required steps to restrict the access granted to connecting systems at the channel level by mapping SSL DNs of the incoming connections to their respective MCAUSER user IDs. From a security perspective, the benefit of using this technique is that the partner has already been authenticated by SSL mutual authentication.

Prior to proceeding with subsequent setup, ensure that the following conditions are true:

- The authorization profiles have been created and configured as described in 11.2, “Authorizing access using the authority context of user IDs” on page 194.
- The queue managers can connect using SSL for mutual authentication as described in 11.4, “Using SSL for mutual authentication” on page 208.

There is an order to pattern matching that determines which CHLAUTH rules take effect. To ensure that the CHLAUTH records are operational, it is a good idea to remove the prior CHLAUTH settings before creating new ones. For more information, see this website:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs14190_.htm

11.5.1 Configuring X.509 DN to MCAUSER mapping

In this section, new CHLAUTH records will be defined to map the X.509 DNs to MCAUSER user IDs. Follow these steps:

1. On the `QMREPOS1` queue manager, define new CHLAUTH records for the cluster receiver channel by using the **`runmqsc`** scripts that are shown in Example 11-39 on page 212.

Example 11-39 CHLAUTH records for queue manager QMREPOS1

```
alt qmgr chlauth(enabled)
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos2,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos2')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm3, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm3')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm4, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm4')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm5, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm5')
refresh security type(ssl)
```

The following explanations refer to Example 11-39:

- The first command ensures that the channel authentication is enabled.
- Subsequent commands configure channel authentication records for the incoming connections to map the X.509 DNs to their respective MCAUSER user IDs.
- The last command forces WebSphere MQ to refresh its security cache to take the new settings into consideration.

Tip: When issuing the **`refresh security`** command, wait for the positive response:

```
AMQ8560: WebSphere MQ security cache refreshed
```

This message indicates that WebSphere MQ has propagated the changes as appropriate and avoids an out-of-sync condition.

2. Update QMREPOS2 by using the **runmqsc** commands shown in Example 11-40.

Example 11-40 CHLAUTH records for queue manager QMREPOS2

```
alt qmgr chlauth(enabled)
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos2,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos2')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm3, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm3')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm4, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm4')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm5, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm5')
refresh security type(ssl)
```

3. Update QM3 by using the **runmqsc** commands shown in Example 11-41.

Example 11-41 CHLAUTH records for queue manager QM3

```
alt qmgr chlauth(enabled)
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos1,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos1')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos2,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos2')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm4, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm4')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm5, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm5')
refresh security type(ssl)
```

4. Update QM4 by using the **runmqsc** commands shown in Example 11-42.

Example 11-42 CHLAUTH records for queue manager QM4

```
alt qmgr chlauth(enabled)
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos1,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos1')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos2,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos2')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm3, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm3')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm5, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm5')refresh security type(ssl)
refresh security type(ssl)
```

5. Update QM5 by using the **runmqsc** commands shown in Example 11-43.

Example 11-43 CHLAUTH records for queue manager QM5

```
alt qmgr chlauth(enabled)
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos1,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos1')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos2,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos2')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm3, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm3')
```

```
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm4, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm4')
refresh security type(ssl)
```

When creating these records, the attribute identifiers in the **sslpeer** option have to be *uppercase*, such as CN, OU, O, and C. If the case is not correct, an error similar to the error shown in Example 11-44 can occur.

Example 11-44 Lowercase is used incorrectly for attribute identifier (cn) in the SSLPEER

```
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('cn=qmrepos1, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qmrepos1')
     1 : set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('cn=qmrepos1,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos1')
AMQ8243: SSLPEER definition wrong.
```

The correct command is shown in Example 11-45.

Example 11-45 Uppercase used as expected for the attribute identifier (CN)

```
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos1, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qmrepos1')
     2 : set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos1,
OU=SA-W216 Residency, O=IBM ITSO, C=US') mcauser('qmrepos1')
AMQ8877: WebSphere MQ channel authentication record set.
```

11.5.2 Validating the settings

Having set up and configured SSL, all the queue managers should now be able to connect to one another securely by means of mutual authentication, that is, the ability to confirm the partner is genuine. Follow these steps:

1. Start the cluster sender channels of the cluster queue managers and confirm that their status is RUNNING.
2. At the receiving queue managers, display the cluster-receiver channel status and confirm that each has the MCAUSER mapped to its respective user ID, For example, on QMREPOS1, display the cluster receiver channel CLUSTER1.QMREPOS1 (Example 11-46).

Example 11-46 Channel status output for a cluster-receiver channel

```
dis chs(CLUSTER1.QMREPOS1) MCAUSER SSLPEER
     6 : DIS CHS(CLUSTER1.QMREPOS1) MCAUSER SSLPEER
AMQ8417: Display Channel Status details.
   CHANNEL(CLUSTER1.QMREPOS1)              CHLTYPE(CLUSRCVR)
   CONNAME(192.168.102.104)                CURRENT
   MCAUSER(qmrepos2)                       RQMNAME(QMREPOS2)
   SSLPEER(SERIALNUMBER=24:3B:0A:5A:00:00:00:00:00:1F,CN=qmrepos2,OU=SA-W216
Residency,O=IBM ITSO,C=US)
   STATUS(RUNNING)                         SUBSTATE(RECEIVE)
```

The channel status output shown in Example 11-46 confirms that the cluster-receiver channel has an SSL/TLS certificate belonging to the QMREPOS2 queue manager and its MCAUSER user ID is correctly set to qmrepos2.

3. Test a use case where the DN is mapped to a non-existent user ID.

   We would logically expect that the channel will not be able to start. The channel should be in a retry state with errors logged in the log file indicating that the user ID is invalid or not authorized.

   However, WebSphere MQ indicates that the channel status is in a `Running` state because, at this stage, the security checks have not been fully carried out and the true status is not determined until the first message is sent. When the first attempt to put a message on the cluster queue that uses this channel fails, it fails with a return code of `2035 - Not authorized`.

11.5.3 Summary

We have configured the cluster environment so that authorized queue managers possessing valid SSL certificates within the cluster can connect to one another by means of SSL mutual authentication. Additionally, the DN of each valid certificate is mapped to a specific authorization role to ensure that each queue manager has sufficient authority to perform a specific task, but no more than that.

11.6 Authorizing access with X.509 and IP address mapping

To circumvent spoofing of a queue manager name by reusing an SSL certificate (which might have been stolen), this technique can be used to ensure that a connection from an authorized remote queue manager associated with an X.509 DN is connecting from a specified address.

Use the **runmqsc** scripts that are shown in Example 11-47 to create the necessary CHLAUTH records for the cluster receiver channel on `QMREPOS1`.

Example 11-47 Create CHLAUTH records for the cluster receiver channel on QMREPOS1

```
alt qmgr chlauth(enabled)
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qmrepos2, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qmrepos2') address('192.168.102.104')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm3, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm3') address('192.168.102.105')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm4, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm4') address('192.168.102.105')
set chlauth(CLUSTER1.QMREPOS1) TYPE(sslpeermap) sslpeer('CN=qm5, OU=SA-W216
Residency, O=IBM ITSO, C=US') mcauser('qm5') address('192.168.102.105')
```

The following explanations refer to Example 11-47:

- The first command ensures that the channel authentication is enabled.
- The subsequent commands configure channel authentication records for the incoming connections to map the X.509 DNs and IP addresses to their respective MCAUSER user IDs. Only when the peer DN and the IP address of the remote queue manager.

The attribute identifiers in the **sslpeer** option have to be *uppercase*, such as `CN`, `OU`, `O`, and `C`.

Use the scripts that are shown in Example 11-47 as a template to update the queue managers `QMREPOS2`, `QM3`, `QM4`, and `QM5`.

11.6.1 Validating the settings

Ensure that all queue managers can connect to one another without errors. You can use a "negative test":

1. Create a new queue manager with the name of an existing queue manager, `QM5` for example, on a server that has a different IP address.
2. Copy the `key.*` files of the original `QM5` queue manager over to the new `QM5` folder, for example, `/var/mqm/qmgrs/QM5/ssl`.
3. Configure the new `QM5` queue manager with a cluster sender/receiver channel that uses SSL just like the original `QM5`.
4. As soon as `QM5` attempts to connect to its designated full-repository queue manager `QMREPOS2`, errors should appear in the error logs on both sides and the request to connect to the cluster from the rogue `QM5` should be rejected.

Example 11-48 shows an error on the rogue `QM5` side when it attempts to connect to the full-repository queue manager.

Example 11-48 Error on the rogue queue manager side when it attempts to connect to the cluster

```
AMQ9506: Message receipt confirmation failed.

EXPLANATION:
Channel 'CLUSTER1.QMREPOS2' has ended because the remote queue manager on host
'wmqsvr4 (192.168.102.104)(1414)' did not accept the last batch of messages.
ACTION:
The error log for the channel at the remote site will contain an explanation of
the failure. Contact the remote Systems Administrator to resolve the problem.
```

Example 11-49 show errors on the full-repository queue manager side, when the rogue `QM5` attempts to connect to the cluster.

Example 11-49 Error on the full-repository queue manager side when rogue QM5 attempts to connect

```
AMQ5653: The user '*NOACCESS' is not defined.

AMQ9557: Queue Manager User ID initialization failed.

EXPLANATION:
The call to initialize the User ID failed with CompCode 2 and Reason 2035.

AMQ9999: Channel 'CLUSTER1.QMREPOS2' to host 'wmqsvr6 (192.168.102.106)' ended
abnormally.
AMQ9999: Channel 'CLUSTER1.QMREPOS2' to host 'wmqsvr6 (192.168.102.106)' ended
abnormally.

EXPLANATION:
The channel program running under process ID 23970 for channel
'CLUSTER1.QMREPOS2' ended abnormally. The host name is 'wmqsvr6
(192.168.102.106)'; in some cases the host name cannot be determined and so is
shown as '????'.
```

This is the sequence of events for the scenario:

1. QM5 attempts to connect to the full-repository queue manager QMREPOS2 by starting its cluster-sender channel CLUSTER1.QMREPOS2.
2. QMREPOS2 receives the request to start its cluster-receiver channel CLUSTER1.QMREPOS2 from QM5.
3. WebSphere MQ performs SSL mutual authentication on both queue managers successfully because both certificates are valid.
4. QMREPOS2 applies the defined CHLAUTH rule for the connection request mandating that both the X.509 DN of QM5 and the IP address of QM5 must match. The rule fails because of the IP address mismatch and therefore the mapping of the MCAUSER user ID to qm5 does not occur.
5. QMREPOS2 attempts to start its cluster-receiver channel using the default MCAUSER user ID of *NOACCESS, which is an invalid user ID. Therefore, the start-up fails.

11.6.2 Summary

We have configured the cluster environment so that authorized queue managers possessing valid SSL certificates can only connect to one another if their respective IP addresses matched that specified in the CHLAUTH rules.

11.7 Considerations for large clusters

The use of CHLAUTH rules to provide fine-grained access control demonstrated so far is manageable for small clusters. For larger clusters, maintaining and managing the rules for individual cluster members and their respective user IDs can be somewhat overwhelming.

As an example, consider a cluster environment comprised of hundreds or thousands of cluster members serving payroll applications across many cities and states, as well as applications providing financial data to the headquarters. The security requirement here is to provide access control in order to maintain a certain level of isolation between business units, but with the least effort required to set up such a topology.

One approach would be to logically group cluster members based on their individual business units and geography locations:

- All payroll cluster members based in Phoenix would be grouped into a logical unit of PHXPAY.
- All payroll cluster members based in Philadelphia would be grouped into a logical unit of PHLPAY.
- All financial cluster members would be grouped into a logical unit of HQFIN.

Using these assumptions, here is an example of some X.509 certificates:

```
CN=webspheremqqm0301, OU=/PHLPAY/, OU=TESTENV, OU=SA-W216 Residency, O=IBM ITSO, C=US
CN=webspheremqqm0302, OU=/PHLPAY/, OU=TESTENV, OU=SA-W216 Residency, O=IBM ITSO, C=US
CN=webspheremqqm0401, OU=/PHXPAY/, OU=TESTENV, OU=SA-W216 Residency, O=IBM ITSO, C=US
CN=webspheremqqm0402, OU=/PHXPAY/, OU=TESTENV, OU=SA-W216 Residency, O=IBM ITSO, C=US
CN=webspheremqqm0501, OU=/HQFIN/, OU=TESTENV, OU=SA-W216 Residency, O=IBM ITSO, C=US
CN=webspheremqqm0502, OU=/HQFIN/, OU=TESTENV, OU=SA-W216 Residency, O=IBM ITSO, C=US
```

A solution would be to check on the OU attributes instead of the CN attribute:

```
set chlauth(CLUSTER1.QM0501) type(sslpeermap) sslpeer('CN=*, OU=/*PAY/,
OU=TESTENV') mcauser('pay')
```

The rule dictates that the queue manager `QM0501` (residing at the headquarters) maps the MCAUSER to user ID "`pay`" for any incoming connection whose OU attribute of the X.509 certificate contains the last three characters of `PAY`. This assumes that the authorization context for user ID pay has been set to allow incoming connections.

11.8 Summary

We have configured a cluster environment with the following security aspects:

- Full-repository queue managers are hosted on dedicated queue managers. These queue managers are there only to support the cluster topology without any application-specific functions or objects.
- Each cluster member queue manager represents a unique security role so that application domains are isolated - an application would be prevented from interfering or compromising the operations of others.
- No unauthenticated or default access is defined. Only enumerated queue managers in the cluster are recognized.
- Each security role allows the required security privilege to perform its duty but not more.

Fine-grained access control and authorization can be applied to a clustered environment. A combination of techniques is used to ensure that only authorized queue managers can connect to one another. Specifically, fine-grained access control is implemented by these controls:

- Mapping queue manager names to the MCAUSER user ID using CHLAUTH channel authentication records
- Authenticating the partner queue manager using SSL mutual authentication
- Mapping an X.509 DN to an MCAUSER user ID using CHLAUTH channel authentication records
- Mapping a DN to an MCAUSER user ID and asserting that the connection has to come from a specified IP address using CHLAUTH channel authentication record
- Providing only sufficient authority for a user ID to perform its role using AUTHREC authority records

There are many other uses of channel authentication records to perform fine-grained access controls that this scenario did not cover, including blocking connections from specific IP addresses and blocking connections from specific user IDs. This chapter focuses primarily on securing a cluster environment and therefore other security aspects are not mentioned in order to focus on this task.

12

Scenario: CRL/OCSP certificate revocation

This chapter describes how to enable certificate revocation checking on IBM WebSphere MQ queue managers. Certificate revocation checking enables the queue manager to reject connection requests where the requestor presents a revoked certificate. The chapter demonstrates the required configuration to enable Certificate Revocation List (CRL) and Online Certificate Status Protocol (OCSP) checking on a queue manager. Troubleshooting tools and techniques that may be helpful are also discussed in this chapter.

This chapter contains the following topics:

- Scenario overview
- Certificate revocation
- Using certificate revocation lists
- Using Online Certificate Status Protocol (OCSP)
- Troubleshooting
- Summary

12.1 Scenario overview

WebSphere MQ supports Secure Sockets Layer (SSL) and Transport Layer Security (TLS) to provide encryption and authentication of remote connections. The authentication aspect is made possible by using X.509 certificates. The identity of the certificate holder, typically an application, server, or human user, is bound cryptographically to the certificate when it is created. In most cases, the certificate is then submitted to a certificate authority (CA) that verifies the identity of the requestor and signs the certificate. The act of signing the certificate binds the CA identity to the certificate along with that of the certificate holder. In this way, the CA can vouch for the identity of the certificate holder and the validity of the certificate.

The certificate contains a private key that the holder must never disclose and a public key that is freely given to any third party who needs to authenticate the certificate. This third party is the relying party and in this scenario is the queue manager to which a connection is requested. When the certificate holder requests a connection, the relying party is able to validate the certificate based on the identities of the holder and CA that are bound to it. Because the identities are cryptographically bound, the relying party has a high level of assurance that the certificate presented is genuine and can make authorization decisions based on the holder's identity as represented by the certificate.

However, when the CA vouches for the certificate, it does so *at a point in time*. There are many cases where it becomes necessary to revoke a valid certificate. It may be that the certificate holder no longer works for the parent company to whom the certificate was issued. A more serious case is that a certificate is compromised. For these cases, some mechanism is necessary to communicate to parties relying on the certificate that it has been revoked.

There are two standard mechanisms for communicating revocation information to relying parties. These are Certificate Revocation Lists (CRL) and Online Certificate Status Protocol (OCSP). A special case of a CRL is an Authority Revocation List (ARL). This is a CRL in which the status of the signer certificates used by the CA may be checked.

WebSphere MQ can be configured for either OCSP checking, CRL/ARL checking, or both. If both methods are enabled, WebSphere MQ uses OCSP for revocation status first for performance reasons. If the revocation status of a certificate is undetermined after the OCSP checking, WebSphere MQ uses CRL checking.

This scenario will demonstrate how an existing WebSphere MQ network, one that is already configured for mutually authenticated SSL, can be enabled to check whether a certificate has been revoked.

12.1.1 CRL design

CRLs and ARLs are provided by the issuing CA as flat files containing the list of all certificates that have been revoked and whose expiry date has not passed. These are usually delivered by the CA over Secure Hypertext Transport Protocol (HTTPS). The downside to CRL checking is the time that it takes to download the entire list of certificates on every connection request. With OCSP, the relying party can perform a lookup against a Lightweight Directory Access Protocol (LDAP) server, specifying the particular certificate of interest.

It is the responsibility of the relying party to implement the LDAP server to respond to these requests. Typically, the LDAP administrator automates a process whereby the CRL is downloaded from the CA's secure web server and loaded into the local LDAP server. This process repeats at least every 12 hours to refresh the CRL data in the LDAP server.

The LDAP servers should be redundant to avoid the potential for a single point of failure and each of the LDAP server instances must contain identical CRL information. There should be workload balancing in place, so failover is automatic.

You need a CRL for each CA that you are using and you must keep them up-to-date. WebSphere MQ maintains a cache of CRLs and ARLs that have been accessed in the preceding 12 hours for better performance.

Figure 12-1 shows an LDAP server providing CRLs to the queue managers.

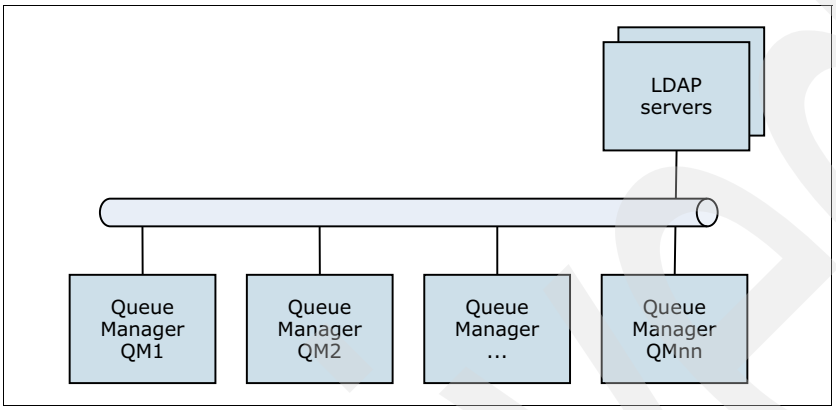


Figure 12-1 LDAP network configuration

The scenario will show how to configure a queue manager with a list of LDAP servers to use for CRL checking and how to activate that list.

Tip: Ensure that the access control list for your LDAP server allows authorized users to read, search, and compare the entries that hold the CRLs and ARLs. WebSphere MQ accesses the LDAP server using the LDAPUSER and LDAPPWD properties of the AUTHINFO object.

12.1.2 OCSP design

Online Certificate Status Protocol (OCSP) was created to address some of the issues with certificate revocation lists (CRLs). With OCSP, the relying party requests the status of a single certificate directly from the CA. Because this method eliminates the requirement to maintain separate LDAP servers with revocation information and refresh those at regular intervals, it is more commonly used.

An extension to the X.509 standard supports OCSP. Certificates using this standard are known as *Extended Validation* or EV Certificates. The EV certificate contains the Uniform Resource Locator (URL) at which the certificate status can be checked. Like everything else in the certificate, the URL is cryptographically bound and any modification to the address will invalidate the certificate.

One issue with OCSP is that when a commercial CA is used, the relying party must make the OCSP call to an external address. Typical security controls on production servers restrict all unnecessary access off the server, especially access through the firewall to and onto the Internet. In the event that a server is compromised, this limits paths of egress for stolen data. The use of OCSP violates this principle by requiring the firewall to allow outbound HTTPS traffic to the Internet.

Although the outbound HTTP access can be restricted by a stateful firewall to a specific URL that is owned and managed by the CA, this creates numerous firewall exceptions. Many companies find it easier to create an internal address, which proxies the request to the CA. For this reason, WebSphere MQ provides the capability to configure the OCSP responder address in the queue manager rather than using the one in the certificate.

If you are using external CAs, it is a preferred practice to have an HTTP proxy or similar device placed in a demilitarized zone (DMZ) to avoid direct network connectivity between your queue manager and the Internet.

WebSphere MQ consults OCSP every time that a Secure Sockets Layer (SSL)/Transport Layer Security (TLS) connection is established if the Authority Information Access (AIA) is present or an AUTHINFO OCSP record is specified. This means that the OCSP responders must have a high level of performance and must be redundant to avoid a single point of failure. This implies that when using an external CA, you need to consider the bandwidth.

Preferred practice: Because revocation checking is performed each time that a connection is established, do not allow WebSphere MQ clients, including Java, Java Message Service (JMS), and .NET applications to poll (MQCONN, MQOPEN, MQGET, MQCLOSE, and MQDISC repeatedly) all the time. Let the application wait for messages for a period of time and then loop to consume them, and not disconnect after each message.

Platform limitations: WebSphere MQ does not currently support OCSP on z/OS and i5/OS.

WebSphere MQ uses OCSP to check that certificate revocation is enabled by default. To check the revocation status of a digital certificate by using OCSP, WebSphere MQ first determines which OCSP responder to contact. It does this in one of two ways:

- Using the AuthorityInfoAccess (AIA) certificate extension in the certificate to be checked (this way is the default behavior)
- Using a URL that is specified in an authentication information object (AUTHINFO) or by the WebSphere MQ client application where it can be specified as an option on an MQCONNX Message Queue Interface (MQI) call

A URL that is specified in an authentication information object or by a client application takes priority over a URL in an AIA certificate extension. This means that you only issue OCSP checks against one CA using an AUTHINFO URL, and the configuration only deals with CA certificates from one vendor. An alternative is to write your own OCSP responder.

Figure 12-2 on page 223 shows an OCSP configuration with one OCSP responder.

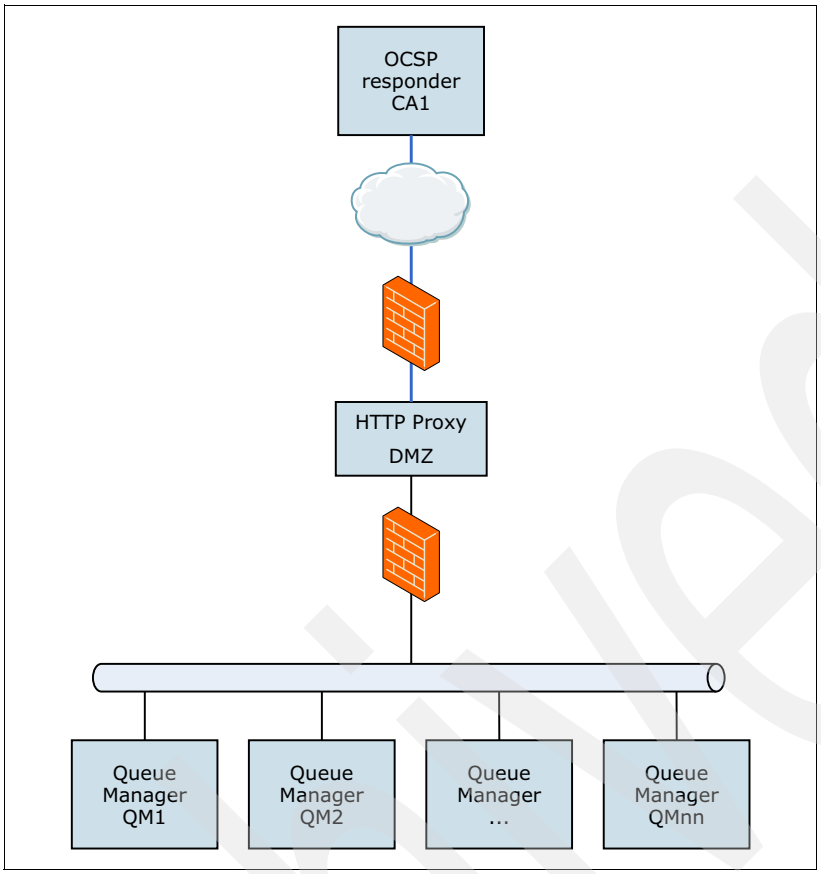


Figure 12-2 OSCP configuration using AUTHINFO entry

The URL of the OCSP responder might lie behind a firewall; if so, configure the firewall so that the OCSP responder can be accessed through it. Alternatively, set up an OCSP proxy server and specify the name of the proxy server by using the SSLHTTPProxyName variable in the SSL stanza. On client systems, you can also specify the name of the proxy server by using the environment variable MQSSLPROXY.

Figure 12-3 on page 224 shows the default AIA configuration, where the queue managers need connectivity to all the CAs from which they have received CA certificates.

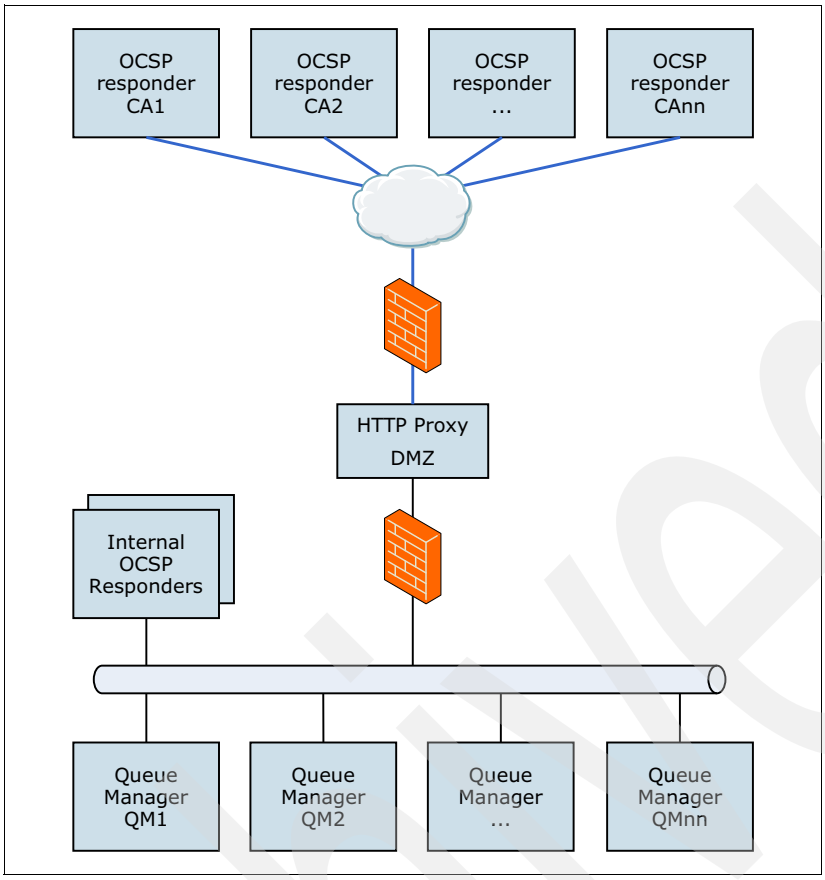


Figure 12-3 OCSP network using default OCSP setup

You can also have an environment with only internal CAs and LDAP servers. This environment is shown in Figure 12-4.

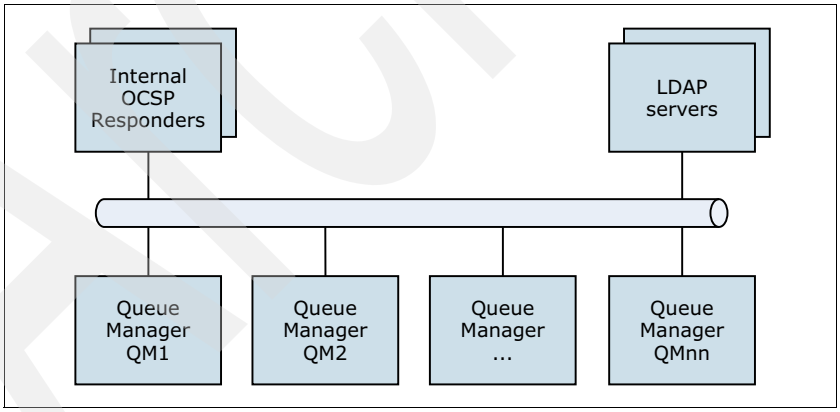


Figure 12-4 Example of internal OCSP responders and LDAP servers

The scenario will show how to configure a queue manager for OCSP checking. For a description of the OCSP environment, see 8.9, “OCSP responder” on page 90.

12.1.3 Prerequisites

The base administrative hardening as described in Chapter 9, “Scenario: WebSphere MQ administration” on page 97 is a prerequisite for this scenario. Queue managers on which administrative access is not locked down cannot be effectively secured in their other aspects, including channels and revocation checking.

The scenario assumes that the queue managers that are involved are configured for mutually authenticated SSL or TLS. For more information about this topic, see the information center topic *Working with SSL and TLS:*

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/zs00140_.htm

SupportPac MS0P is referenced in this scenario. This SupportPac includes a plug-in for WebSphere MQ Explorer that parses event messages into human-readable format and presents them on a graphical user interface (GUI). Other SupportPacs and third-party tools are available to perform the same function. SupportPac MS0P and many others can be found at the main SupportPacs page:

http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg27007197

An LDAP server implemented using Tivoli Directory Server is installed and running. The server is used to serve the CRLs. For a description of the LDAP server that is used in this scenario, see 8.10, “LDAP server to host CRLs” on page 90.

12.1.4 Certificate authorities that are used in this scenario

A description of the certificate authorities that are used in this scenario follows. For more information about the CA for the scenario, see 8.8, “Certificate authorities” on page 89.

One of the roles of the CA is to check that the requester of a certificate is authorized to request the certificate and that the certificate contains the right hierarchy of information in the certificate distinguished name (DN). A CA is required to validate at least the company and country information to protect companies against fraud. The internal organizational and functional information should be validated by the IT security department of the requesting organization. Rigorous verification and authentication of requesting parties is critical to certificate-based authentication.

The scenario was developed using an internal CA with a root certificate and an intermediate signer certificate. An illustration of the certificates that are used is provided in Figure 12-5 on page 226.

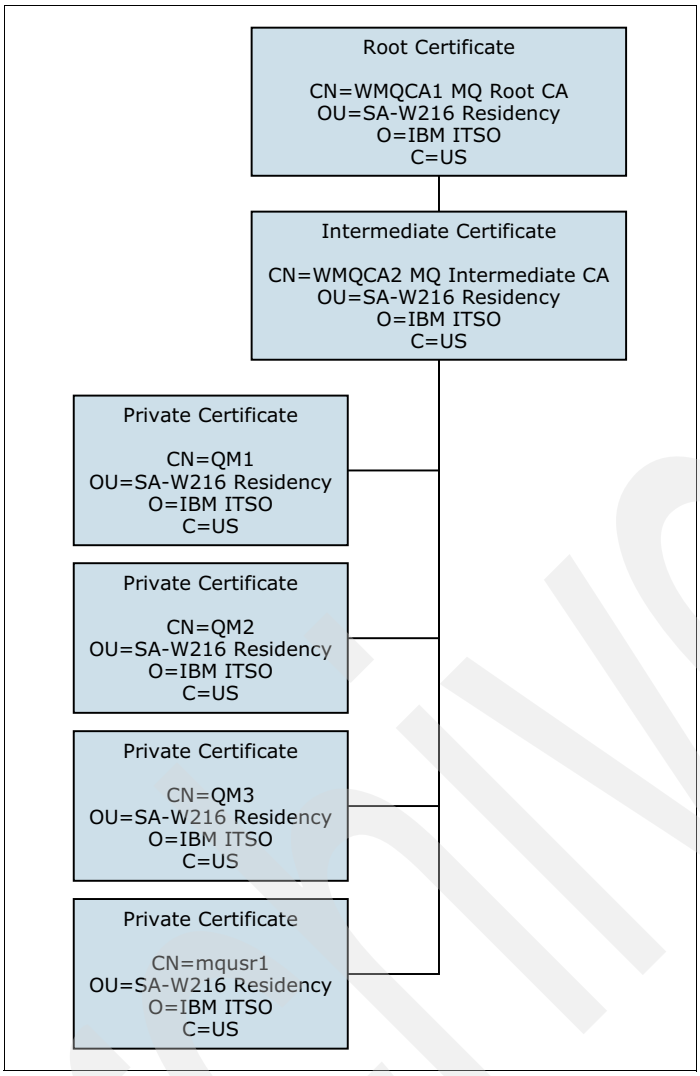


Figure 12-5 Certificate hierarchy used in examples

12.2 Certificate revocation

When using certificates, you need to be able to check the validity of a certificate, similar to the way you would check the validity of a username and password. When a queue manager receives a certificate, the only thing that the queue manager can check is the validity period and certificate chain. A second check is needed to ensure that the certificate has not been terminated, or revoked.

If you terminate a self-signed certificate, you remove it from the keystore that belongs to the queue manager and the problem is solved. When using CA certificates, only the root, intermediate certificates, and the queue manager's private certificate are stored in each queue manager's keystore. This means that help is needed to determine if the presented certificate is still valid.

Important: Only certificates that are needed should be kept in the keystore. Remove any others.

12.3 Using certificate revocation lists

This section will describe how to configure a queue manager for certificate revocation checking using CRL and LDAP. It will also describe how to turn off CRL certificate revocation checking.

12.3.1 Configuring CRL in WebSphere MQ

To implement CRL using LDAP, you have to tell WebSphere MQ how to reach the LDAP server. This is done using the AUTHINFO object in the queue manager. You can define up to 10 LDAP servers, for WebSphere MQ to check for CRLs.

z/OS only: All LDAP servers must be accessed using the same user ID and password. The user ID and password that are used are those specified in the first AUTHINFO object in the name list.

On all platforms: The user ID and password are sent to the LDAP server unencrypted.

Example 12-1 shows the commands that are used to configure the queue manager to perform the LDAP check.

Example 12-1 Configure queue manager to perform LDAP check

```
$ runmqsc QM1
   DEFINE AUTHINFO(WMQCA2.CRL.1) AUTHTYPE(CRLLDAP) +
   DESCR('LDAP lookup of CRLs for wmqca2') CONNAME('wmqldap1') +
   LDAPUSER('cn=mqmbind,ou=bindusers') LDAPPWD('mqmpass') REPLACE

   * Define a namelist referring the LDAP servers to check against
   DEFINE NAMELIST(AUTH.SERVERS) DESCR('list of available revocation info') +
   NAMES(WMQCA2.CRL.1) REPLACE

   * Activate CRL namelist in the queue manager
   ALTER QMGR SSLCRLNL(AUTH.SERVERS)

   end
```

WebSphere MQ will check the LDAP servers in the order that you specify them in the SSLCRLNL name list until it finds an available server. If a CRL is not available, the channel will not start since it cannot be sure that the certificate has not been revoked. This means that the LDAP servers need the same level of high availability as that of your queue managers, if not higher.

12.3.2 Turning off CRL checking

In the event of a problem, for example, the LDAP servers are not available, you can turn CRL checking off quickly by removing `SSLCRLNL` from the queue manager object. Example 12-2 on page 228 shows how to remove `SSLCRLNL` from the queue manager object.

Example 12-2 Configure queue manager to perform LDAP check

```
$ runmqsc QM1
   * Disable CRL namelist in the queue manager
   ALTER QMGR SSLCRLNL(' ')

   * Activate the change
   REFRESH SECURITY TYPE(SSL)

   end
```

Warning: Use this method only as a temporary fix until the root cause can be found.

12.3.3 For more information

For more information, see these resources:

- *Setting up LDAP servers* in the WebSphere MQ Information Center for a description of how to set up connectivity from WebSphere MQ to an LDAP server:

  http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/sy12680_.htm

- RFC3280, which is provided by The Internet Engineering Task Force (IETF) on the following website, for a complete description of the CRL protocol:

  http://www.ietf.org/rfc/rfc3280.txt

12.4 Using Online Certificate Status Protocol (OCSP)

This section will describe how to configure a queue manager for certificate revocation checking using OCSP. It will also describe how to make OCSP checking optional.

12.4.1 Configuring OCSP in WebSphere MQ

This section contains a description of the OCSP configuration that is used in this scenario.

OCSP Extension checks are controlled by settings in the queue manager configuration file (`qm.ini`) or the client configuration file (`mqclient.ini`). The settings are found in the SSL and TLS stanza of the queue manager configuration file and in the SSL stanza of the client configuration file.

- The OCSPCheckExtensions setting specifies whether channels on the queue manager check with OCSP servers that are named in the AuthorityInfoAccess certificate extensions. The valid settings are either YES or NO:
  - YES: SSL and TLS channels try to check OCSP servers to determine whether a digital certificate is revoked. This value is the default.
  - NO: SSL and TLS channels do not try to check OCSP servers. This setting should be used with caution and only in test environments.

- The OCSPAuthentication setting determines what happens when OCSP checking is enabled but the revocation status cannot be determined from an OCSP server:
  - REQUIRED: Failure to determine the revocation status causes the connection to be closed with an error. This value is the default.
  - WARN: Failure to determine the revocation status causes a warning message to be written in the queue manager error log, but the connection is allowed to proceed.
  - OPTIONAL: Failure to determine the revocation status allows the connection to proceed silently. No warnings or errors are given.

Important: Specify all settings rather than relying on defaults. Not only do defaults change over time but the value could be misinterpreted by someone. Specifying each setting ensures that the intended meaning is clear.

Figure 12-6 shows the OCSP settings as seen in the MQ Explorer.

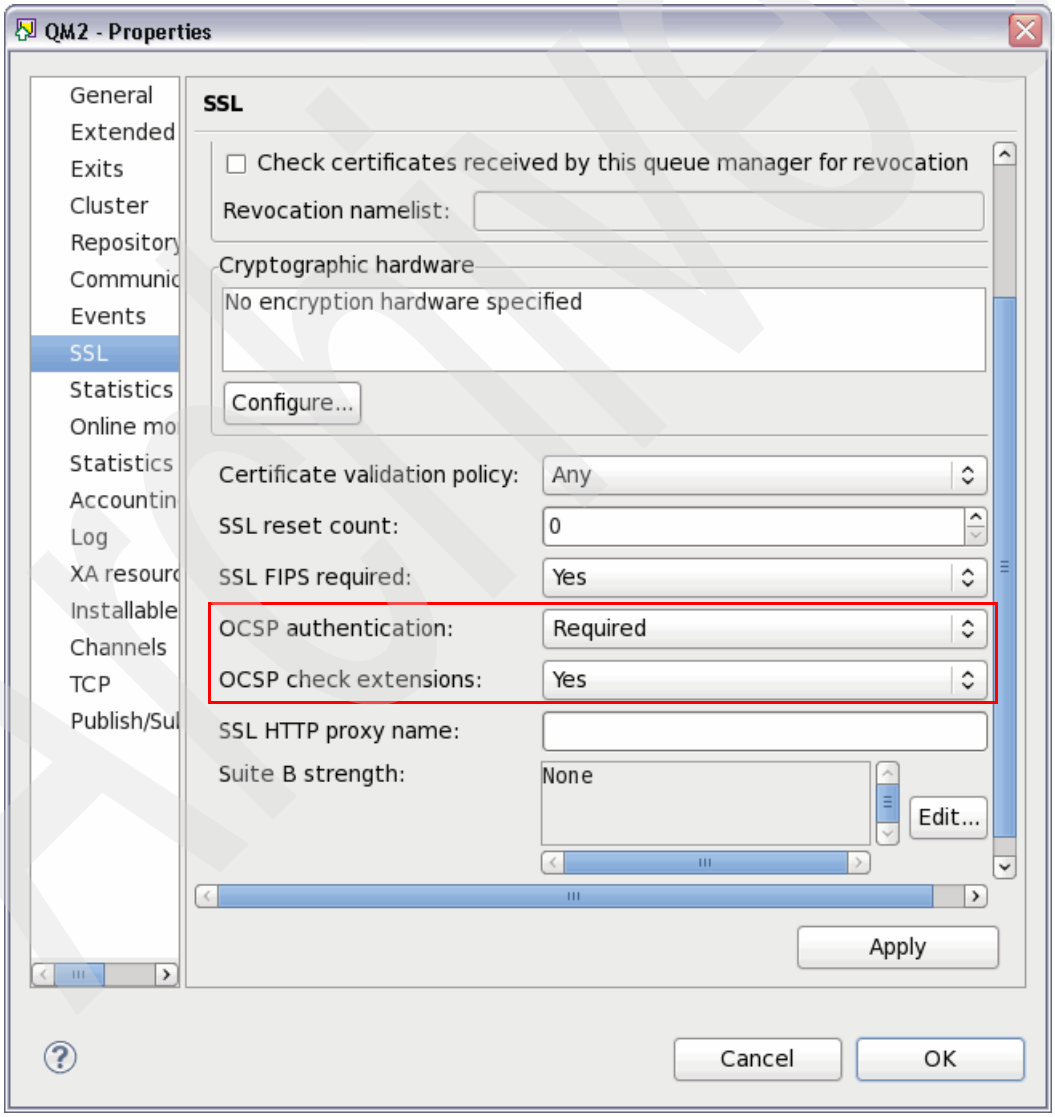


Figure 12-6 Setting OCSP properties using MQ Explorer

Example 12-3 shows `qm.ini` with the OCSP authentication enabled and extended attribute checking enabled.

Example 12-3 A qm.ini with OCSP extended attribute checking enabled

```
#*******************************************************************#
#* Module Name: qm.ini                                             *#
#* Type       : WebSphere MQ queue manager configuration file      *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                 *#
#*******************************************************************#
#* Notes      :                                                    *#
#* 1) This file defines the configuration of the queue manager     *#
#*                                                                 *#
#*******************************************************************#
ExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64
#*                                                                 *#
#*                                                                 *#
Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=4096
   LogType=CIRCULAR
   LogBufferPages=0
   LogPath=/var/mqm/log/QM2/
   LogWriteIntegrity=TripleWrite
Service:
   Name=AuthorizationService
   EntryPoints=14
ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=amqzfu
   ComponentDataSize=0
SSL:
   OCSPCheckExtensions=YES
   SSLFipsRequired=YES
   OCSPAuthentication=REQUIRED
```

Example 12-4 shows the OCSP extended attribute set in `mqclient.ini`.

Example 12-4 A mqclient.ini with OCSP extended attribute checking enabled

```
#*************************************************************************#
#* Module name : mqclient.ini
#* Type        : WebSphere MQ client configuration file
#* Function    : Define the client configuration
#*
#*************************************************************************#

ClientExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64
SSL:
```

```
   OCSPCheckExtensions=YES
   SSLFipsRequired=YES
   OCSPAuthentication=REQUIRED
```

Important: When using AIA extensions, make sure that you know the OCSP servers that will be named and ensure that they are accessible over the network.

12.4.2 Making OCSP checking optional

If you lose connectivity to your OCSP provider, you will not be able to start new connections. A workaround for this situation is to set the OCSPAuthentication setting to OPTIONAL. With this setting, OCSP checking will still occur.

- If REVOKED is returned, the channel will not start and `AMQ9716 (Remote SSL certificate revocation status check failed)` is issued.
- If GOOD is returned, the channel is allowed to start.
- If UNKNOWN is returned, the channel is allowed to start and `AMQ9716` is not issued.

Example 12-5 shows how to change this setting for UNIX and Linux systems in `qm.ini`.

Example 12-5 Change OCSP Authentication setting on UNIX and Linux systems

```
SSL:
   OCSPAuthentication=OPTIONAL
```

To change the setting for a queue manager on the Microsoft Windows platform, run the following command.

Example 12-6 Change OCSP Authentication on Windows systems

```
amqmdain reg YourQMgrName -c add -s SSL -v OCSPAuthentication=OPTIONAL
```

The OCSPAuthentication setting is also applicable to clients and can also be set in the client ini file (`mqclient.ini`). Changes to the client OCSPAuthentication setting will take effect after the application is restarted.

You need to issue a **`REFRESH SECURITY TYPE(SSL)`** command (or the GUI/Programmable Command Format (PCF) equivalent) in order for the change of SSL settings to take effect. Example 12-7 shows how to issue the command.

Example 12-7 Refresh SSL command

```
$ runmqsc QM1
   * Activate the change
   REFRESH SECURITY TYPE(SSL)

   end
```

Warning: OCSP gives your SSL-protected channels additional protection by refusing connections that present revoked certificates. After the cause of the problem has been addressed, ensure that you re-enable OCSP enforcement. Use this method only as a temporary fix until the root cause can be found.

12.4.3 For more information

For a complete description of the OCSP protocol, see RFC2560, which is provided by The Internet Engineering Task Force (IETF), on the following website:

http://www.ietf.org/rfc/rfc2560.txt

12.5 Troubleshooting

This section provides some troubleshooting tips.

12.5.1 Security troubleshooting

In the event that you are having problems with your certificate revocation checking, be sure to start by checking all relevant logs. In addition to the WebSphere MQ logs, check those logs that are provided by the operating system:

- Windows:
  - *<install path>*\errors\AMQERR01.LOG, AMQERR02.LOG, and AMQERR03.LOG
  - *<install path>*\qmgrs*<qmgr name>*\errors\AMQERR01.LOG, AMQERR02.LOG, and AMQERR03.LOG
  - Windows event viewer
- Linux/UNIX:
  - *<install path>*/errors/AMQERR01.LOG, AMQERR02.LOG, and AMQERR03.LOG
  - *<install path>*/qmgrs/*<qmgr name>*/errors/AMQERR01.LOG, AMQERR02.LOG, and AMQERR03.LOG
- z/OS:

  Use the spool display facility (SDSF) or a similar tool to look for information that is relevant to the error in *<qmgr name>*MSTR and *<qmgr name>*CHIN.

Error messages will often have helpful information that is included. Example 12-8 shows an example AMQERR01 log on a UNIX operating system. The error message indicates that an SSL DN does not match the SSL peer name for the channel. The action portion of the message provides help in determining and fixing the problem.

Example 12-8 Example from AMQERR01.LOG on UNIX

```
----------------------------------------------------------------------------
07/12/2012 11:33:47 AM - Process(14215.1) User(mqm) Program(runmqchl)
                    Host(wmqsvr2.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(QM2)

AMQ9636: SSL distinguished name does not match peer name, channel 'QM2.QM1'.

EXPLANATION:
The distinguished name,
'SERIALNUMBER=33:DD:C2:26:00:00:00:00:00:2C,CN=QM1,OU=SA-W216 Residency,O=IBM
ITSO,C=US', contained in the SSL certificate for the remote end of the channel
does not match the local SSL peer name for channel 'QM2.QM1'. The distinguished
name at the remote host '9.42.170.149 (9.42.170.149)(14130)' must match the
peer name specified (which can be generic) before the channel can be started.
```

```
ACTION:
If this remote system should be allowed to connect, either change the SSL peer
name specification for the local channel so that it matches the distinguished
name in the SSL certificate for the remote end of the channel, or obtain the
correct certificate for the remote end of the channel. Restart the channel.
```

The following sections contain other examples of error messages and possible solutions.

Error: missing CA certificates in the keyring

In this example (Example 12-9), output from CSQ1CHIN on a z/OS system shows an error:

"CSQX633E -CSQ1 CSQXRCTL SSL certificate for remote channel CSQ1.QM2 failed local check, connection (9.42.171.232)."

Example 12-9 Example of job log from CSQ1CHIN (channel initiator)

```
Display  Filter  View  Print  Options  Search  Help
 -------------------------------------------------------------------------------
 SDSF OUTPUT DISPLAY CSQ1CHIN STC29234  DSID     2 LINE 122     COLUMNS 21- 100
 COMMAND INPUT ===>                                            SCROLL ===> CSR
+CSQX500I -CSQ1 CSQXRCTL Channel CSQ1.QM2 started
+CSQX633E -CSQ1 CSQXRCTL SSL certificate for remote channel CSQ1.QM2  037
 failed local check, connection (9.42.171.232)
+CSQX599E -CSQ1 CSQXRCTL Channel CSQ1.QM2 ended abnormally
+CSQX618I -CSQ1 CSQXRSSL SSL key repository refresh started
+CSQX619I -CSQ1 CSQXRSSL SSL key repository refresh processed
+CSQX500I -CSQ1 CSQXRESP Channel SYSTEM.ADMIN.SVRCONN started  058
 connection 9.146.180.54
+CSQX500I -CSQ1 CSQXRCTL Channel CSQ1.QM2 started
+CSQX632I -CSQ1 CSQXRCTL SSL certificate has no associated user ID,  060
 remote channel CSQ1.QM2
 connection (9.42.171.232)
 - channel initiator user ID used
+CSQX545I -CSQ1 CSQXRCTL Channel CSQ1.QM2 closing because disconnect  061
 interval expired
+CSQX501I -CSQ1 CSQXRCTL Channel CSQ1.QM2 no longer active
******************************** BOTTOM OF DATA ********************************
```

The error message is caused by a missing certificate. Example 12-10 shows how to display the certificates in the keyring.

Example 12-10 Display certificates connected to keyring IBM1RING

```
RACDCERT ID(STC) LISTRING(IBM1RING)

Digital ring information for user STC:
Ring:
       >IBM1RING<
   Certificate Label Name             Cert Owner     USAGE      DEFAULT
   --------------------------------   ------------   --------   -------
   ibmWebSphereMQIBM1                 ID(STC)        PERSONAL     NO
   ibmWebSphereMQCSQ1                 ID(STC)        PERSONAL     NO
READY
```

The problem in this example is that the root certificates are missing. The action to take is to connect the root certificates to keyring IBM1RING (“Connecting certificates to a keyring” on page 310). Then, issue a **REFRESH SECURITY TYPE(SSL)** command (or the GUI/PCF equivalent) in order for the change of SSL settings to take effect, as shown in Example 12-11.

Example 12-11 Refresh SSL operator command

```
-CSQ1 REFRESH SECURITY TYPE(SSL)
```

Example 12-12 shows the certificates in the keyring after the root CA certificates were added.

Example 12-12 Certificates connected to keyring IBM1RING after adding root certificates

```
RACDCERT ID(STC) LISTRING(IBM1RING)

Digital ring information for user STC:

   Ring:
       >IBM1RING<
   Certificate Label Name             Cert Owner     USAGE      DEFAULT
   -------------------------------    ------------   --------   -------
   WMQCA1 MQ Root CA                  CERTAUTH       CERTAUTH     NO
   WMQCA2 MQ Intermediate CA          CERTAUTH       CERTAUTH     NO
   ibmWebSphereMQIBM1                 ID(STC)        PERSONAL     NO
   ibmWebSphereMQCSQ1                 ID(STC)        PERSONAL     NO
READY
```

Error: SSL CRL namelist is empty or wrong type

Example 12-13 shows an error in the log that indicates a problem with CRL checking on z/OS due to parameter problems in the LDAP configuration.

Example 12-13 Error caused by missing a NAMELIST

```
+CSQX162E -CSQ1 CSQXGIMP SSL CRL namelist is empty or wrong type
```

This error was caused by missing NLTYPE(AUTHINFO) on the NAMELIST entry. NLTYPE is required on z/OS for AUTHINFO name lists.

Example 12-14 Name list definition, including NLTYPE

```
DEFINE NAMELIST(AUTH.SERVERS) +
   DESCR('List of available revocation CRL info') +
   NLTYPE(AUTHINFO) +
   NAMES(WMQCA2.CRL) REPLACE
```

Error: LDAP server is unavailable

Example 12-15 shows an error that indicates that the LDAP server is unavailable.

Example 12-15 LDAP server unavailable

```
CSQX666E -CSQ1 CSQXRCTL LDAP server unavailable for channel CSQ1.QM2
```

z/OS requires a host name to be specified for the CONNAME parameter in the AUTHINFO definition, which was missing in this case. Example 12-16 shows the correct definition.

Example 12-16 AUTHINFO command that works

```
DEFINE AUTHINFO(WMQCA2.CRL) descr('ldap lookup of CRLs for wmqca2') +
   AUTHTYPE(CRLLDAP) CONNAME('wmqca2.itso.ral.ibm.com') +
   LDAPUSER('cn=mqmbind,ou=bindusers') +
   LDAPPWD('mqmpass') REPLACE
```

12.5.2 Event messages

WebSphere MQ helps you understand security issues that occur by posting event messages to SYSTEM.ADMIN queues, if enabled. The following command is used to switch on authority and SSL events on a queue manager:

```
ALTER QMGR SSLEV(ENABLED) AUTHOREV(ENABLED)
```

WebSphere MQ will put report messages on security events that relate to channels into SYSTEM.ADMIN.CHANNEL.EVENT. Queue-related security events are put on SYSTEM.ADMIN.QMGR.EVENT.

The best way to display the content of SYSTEM.ADMIN.QMGR.EVENT is either with IBM SupportPac MO71 (WebSphere MQ for Windows - GUI Administrator) or MS0P (WebSphere MQ Explorer - Configuration and Display Extension Plug-ins).

Displaying event messages with MO71

SupportPac MO71 is available from the IBM SupportPac website:

http://www-01.ibm.com/support/docview.wss?uid=swg24000142

Figure 12-7 on page 236 shows an MQRC_NOT_AUTHORIZED event that is related to a queue for user `mqcust2` in SYSTEM.ADMIN.QMGR.EVENT using the MO71GUI Administrator.

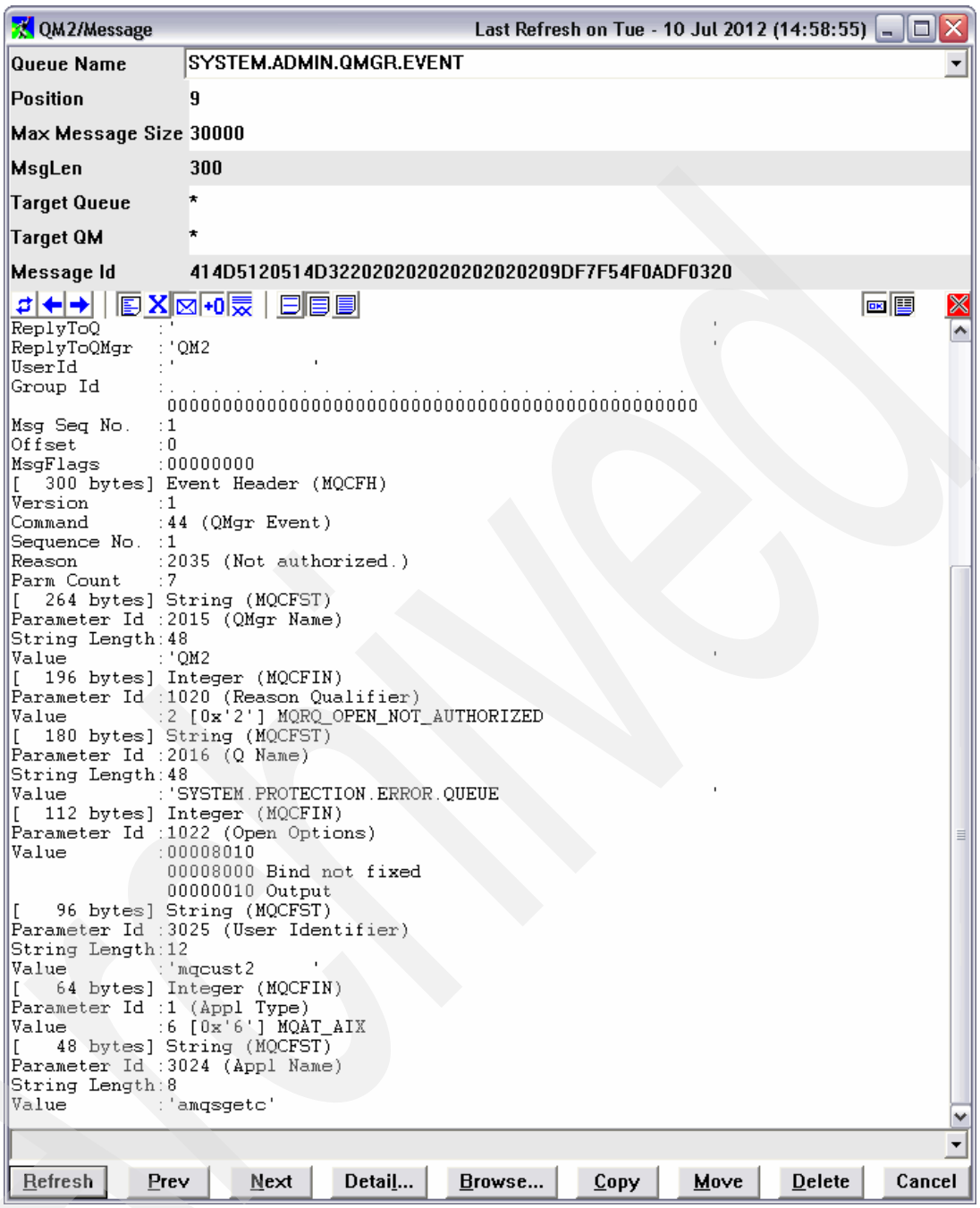


Figure 12-7 Example of an MQRC_NOT_AUTHORIZED event related to a queue for user mqcust2

Displaying event messages with MS0P

SupportPac MS0P is available from the IBM SupportPac website:

http://www-01.ibm.com/support/docview.wss?uid=swg24011617

Figure 12-8 on page 237 shows an MQRC_NOT_AUTHORIZED event that is related to a queue for user `mqcust2`.

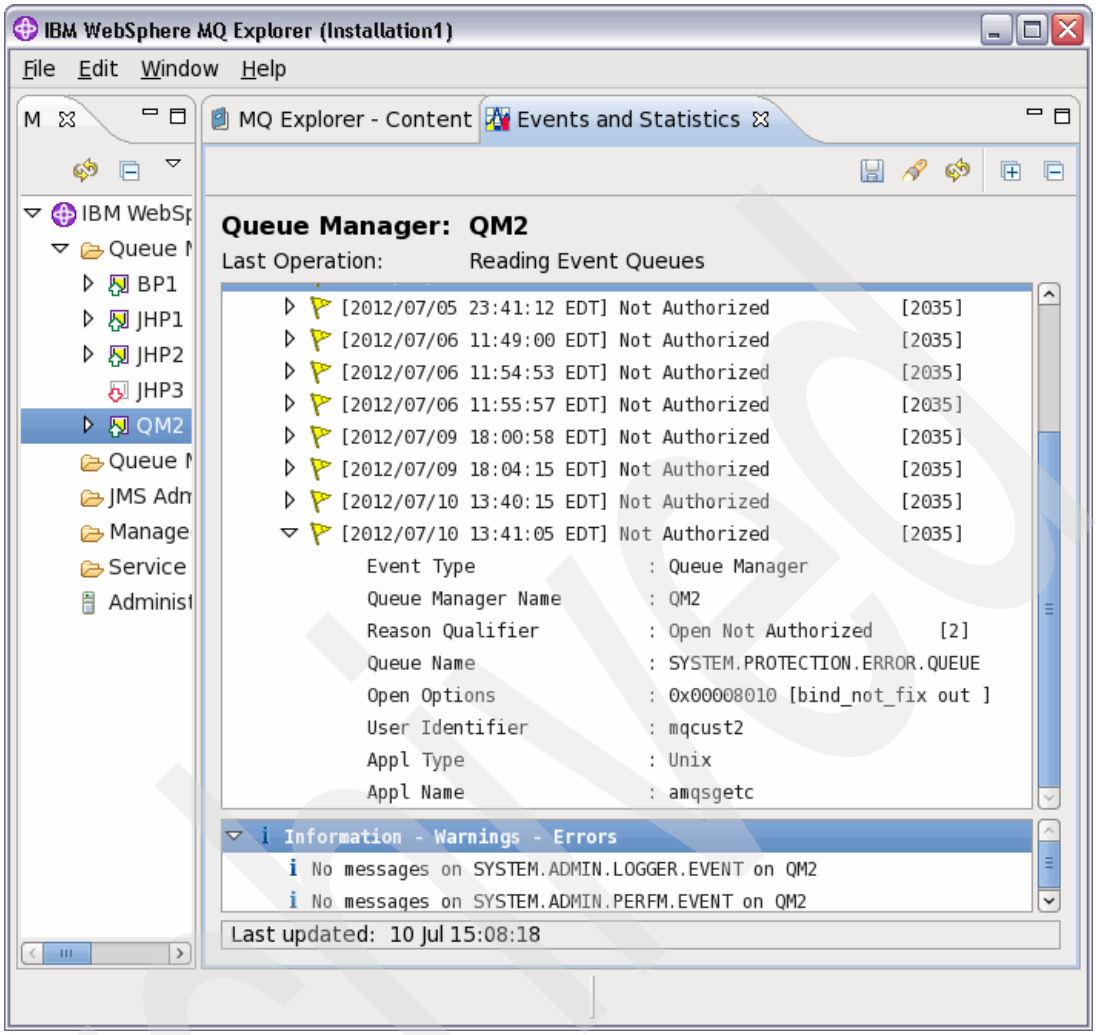


Figure 12-8 Example of an MQRC_NOT_AUTHORIZED event related to a queue for user mqcust2

With IBM SupportPac MS0P, you can browse log files and First Failure Support Technology (FFST), as shown in Figure 12-9.

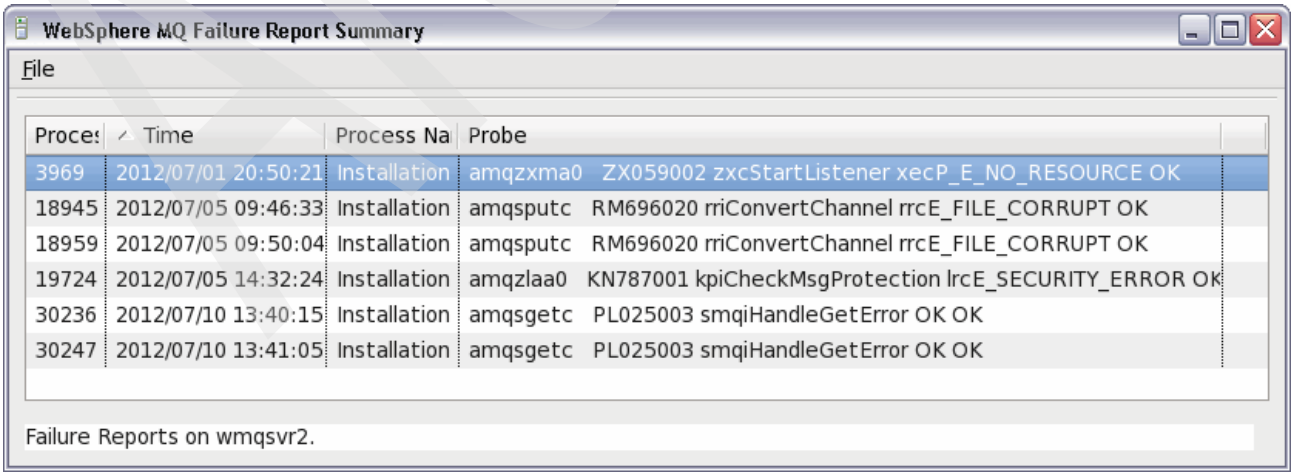


Figure 12-9 Example of FFST overview, including headings from FFST reports

12.5.3 SSL troubleshooting

SSL troubleshooting can be difficult because the cryptographic functions in Global Secure ToolKit (GSKit) are protected from publication to protect your security. You can find some help about GSKit reason codes in the programming manual for z/OS: *z/OS V1R12.0 Cryptographic Services System Secure Sockets Layer Programming z/OS V1R12.0*, which is available at this website:

http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=%2Fcom.ibm.zos.r12.gska100%2Fgska1a80.htm

Example 12-17 shows an error that is reported from GSKit, which is invoked by WebSphere MQ.

Example 12-17 Error reported from GSKit

```
----- amqrmrsa.c : 889 ------------------------------------------------------
07/09/2012 11:27:09 PM - Process(23345.114) User(mqm) Program(amqrmppa)
                    Host(wmqsvr2.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(QM2)

AMQ9620: Internal error on call to SSL function on channel '????' to host
'wmqsvr2-m (192.168.101.102)'.

EXPLANATION:
An error indicating a software problem was returned from a function which is
used to provide SSL or TLS support. The error code returned was '12'. The
function call was 'gsk_secure_soc_init'.

The channel is '????'; in some cases its name cannot be determined and so is
shown as '????'. The channel did not start.

The remote host name is 'wmqsvr2-m (192.168.101.102)'.
ACTION:
Collect the items listed in the 'Problem determination' section of the System
Administration manual and use either the MQ Support site:
http://www.ibm.com/software/integration/wmq/support/, or IBM Support Assistant
(ISA): http://www.ibm.com/software/support/isa/, to see whether a solution is
already available.  If you are unable to find a match, contact your IBM support
center.
```

When diagnosing SSL problems that are associated with starting interconnecting queue managers, try to start with as simple a scenario as possible and then enhance it until you run into the problem:

1. Start the channels without SSL to ensure that the network path is in place.
2. Set the agreed-upon SSL CipherSpec to verify that the certificates are in place and that you can start the channel without problems.
3. Display the certificate's distinguished name with the MQI command **`display channel status`** to see if it is what you expect.
4. Change the SSLPEER settings to the agreed value.

12.6 Summary

In this chapter, you have learned about certification revocation and how to implement it in WebSphere MQ by using either OCSP or LDAP, availability and performance considerations, and troubleshooting OCSP/LDAP issues.

13

Scenario: End-to-end security using WebSphere MQ AMS

This chapter presents a scenario showing how to enable IBM WebSphere MQ Advanced Message Security (AMS) on an existing point-to-point messaging interface between WebSphere MQ client applications. This configuration protects messages from the point in time that the sender executes the `put` API call to the point in time that the message is received by the recipient after the `get` API call.

This scenario is often referred to as end-to-end security because messages are protected while traversing the network, while at rest on the queues, and at all points in between. When messages are encrypted and the decryption keys do not reside on the queue manager servers, there is no possibility that the messages in plaintext can leak out in dumps, log files, memory inspection, or any other method while in custody of the queue manager. For that reason, this scenario is useful as part of a regulatory compliance program or other high-security environments.

This chapter contains the following topics:

- Scenario overview
- Configuring for first use
- Exchanging signed messages
- Exchanging encrypted messages
- Summary
- Further considerations

13.1 Scenario overview

WebSphere MQ Advanced Message Security (AMS) extends WebSphere MQ with security controls that function on individual messages. This is often referred to as *message-level security* to differentiate it from connection-level security, which operates based on authenticated queue manager connections. Because message-level security as implemented in WebSphere MQ AMS is performed in the application's runtime environment rather than by the queue manager, this is also referred to as *application-level security*.

WebSphere MQ AMS signs and encrypts messages using public and private keys and standard X.509 certificates. Signed messages provide an integrity checking mechanism and encrypted messages provide privacy between the sender and intended recipients. Because the identity of the sender, and potentially also the recipients, is cryptographically bound to the message, it is possible to enforce policies as to who is authorized to send or receive the messages. WebSphere MQ AMS provides command-line and graphical tooling to define and manage policies.

This chapter is intended to illustrate the common scenario where an existing point-to-point messaging interface between two applications is upgraded to enable WebSphere MQ AMS.

For more information regarding WebSphere MQ AMS, see the product information center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/as00000_.htm

13.1.1 Scenario design

The diagram in Figure 13-1 shows the topology of the scenario.

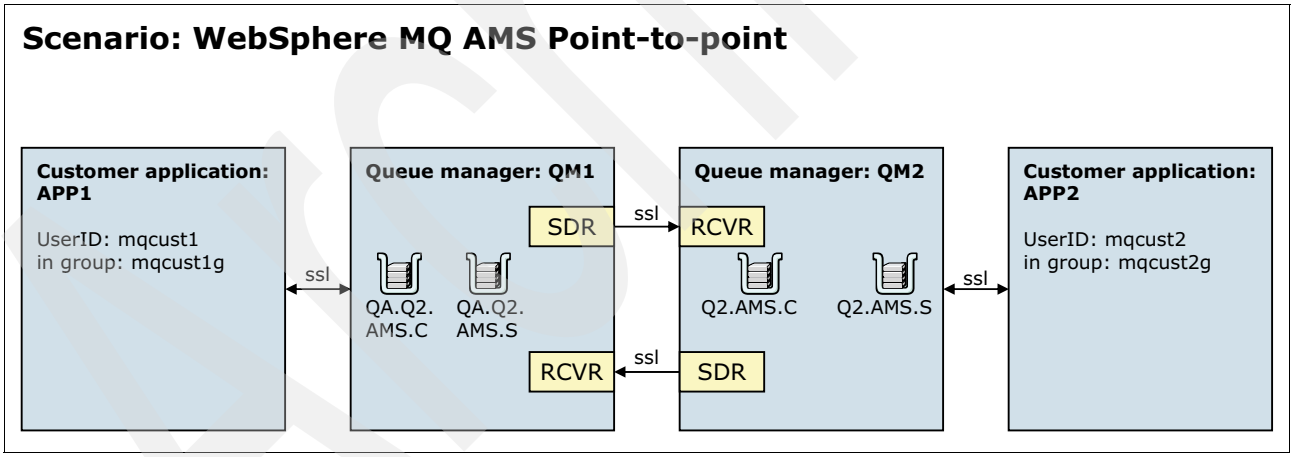


Figure 13-1 Applications and queue managers in WebSphere MQ AMS point-to-point scenario

The scenario uses WebSphere MQ version 7.5, which embeds WebSphere MQ AMS, formerly a separate product. The example will demonstrate the steps to provision certificates and enable interception at the client applications, as well as the steps to configure the queue manager and define policy on the queues that are involved.

This scenario also deliberately uses a configuration in which the two applications are *not* connected to the same queue manager. This provides the opportunity to illustrate how matching protection policies are defined for message senders and recipients across the messaging network. The queue managers can be linked by distributed channels, a WebSphere MQ cluster, intermediate queue managers, or any other valid WebSphere MQ architecture; it does not affect the WebSphere MQ AMS configuration.

The applications in this scenario connect to WebSphere MQ by using client channels. One important reason for this type of configuration is that the client application certificates do not need to exist on the queue manager's host server, and therefore not even the WebSphere MQ administrator can see or tamper with the message. Messages can only be consumed by holders of valid certificates who are named on the protection policy as authorized recipients when the message is created.

The applications in this scenario use service accounts named `mqcust1` and `mqcust2`. Various configuration elements, such as keystores and certificates belonging to each application, also use these names.

The certificates that are used are signed by the certificate authority (CA) that is described in Chapter 8, "Scenario preparation" on page 83. Self-signed certificates could be used, however to do so requires the exchange of the public keys before the applications can exchange signed or encrypted messages. With CA-signed certificates, the exchange of public keys is only required if messages are to be encrypted. Also, if using self-signed certificates, be sure to inspect them and ensure that the attribute `IsCA` is false. If the attribute is true, the self-signed certificate can act as a root CA and forge certificates. When creating self-signed certificates using Global Secure ToolKit (GSKit) 8, which is the set of crypto tools supplied as part of WebSphere MQ, the **`-ca false`** parameter is required to create a non-root personal certificate that is appropriate for this type of usage.

13.1.2 Prerequisites

This scenario assumes that two WebSphere MQ applications are able to exchange messages over WebSphere MQ before the scenario begins. In this case, the applications involved are connected to different queue managers that use sender and receiver channels to exchange messages. The topology is incidental to the example. It was chosen to illustrate distributed policy rules but is applicable to any network topology where two applications can exchange messages, whether that is a single queue manager, two queue managers as shown here, or a multi-hop topology.

Since WebSphere MQ V7.5 is used, there is no additional installation step for WebSphere MQ AMS as there would be in prior versions of WebSphere MQ. If using WebSphere MQ V7.1, refer to the information center for installation instructions. After installation, the remaining steps are identical in either version.

WebSphere MQ AMS V7.0.1 installation instructions can be found in the information center:

`http://publib.boulder.ibm.com/infocenter/mqams/v7r0m1/index.jsp`

The scenario also assumes that basic administrative hardening has been performed on the queue managers that are used in the scenario as described in Chapter 9, "Scenario: WebSphere MQ administration" on page 97. WebSphere MQ AMS extends the security in base WebSphere MQ but does not replace it. In particular, this scenario requires that the WebSphere MQ administrator has secure access to the queue managers using WebSphere MQ Explorer in order to administer the WebSphere MQ AMS protection policies.

13.2 Configuring for first use

This portion of the scenario consists of the required installation and configuration to use WebSphere MQ AMS. Since the applications and queue managers are already exchanging messages, the only task in WebSphere MQ AMS V7.5 is to provision keystores and keys for the applications. Some additional configuration is required for earlier versions and this is pointed out where relevant.

13.2.1 Queue manager configuration

Because the scenario is implemented using WebSphere MQ V7.5, there are no tasks to prepare the queue manager.

In versions prior to V7.5, it is necessary to define SYSTEM.PROTECTION.POLICY.QUEUE and SYSTEM.PROTECTION.ERROR.QUEUE before enabling WebSphere MQ AMS on the client applications. These are ordinary local queues and versions requiring this step include a script to define them. In addition, you must enable AMS with the following command:

```
cfgmqs -enable -client
```

13.2.2 Authorizations

The applications that have been exchanging messages are already authorized to the queues that are used for data. In order to use WebSphere MQ AMS, they also must be authorized to the policy and exception queues. It is important that the applications are not authorized to put messages to the policy queue. Typically, applications are authorized to put messages on the exception queue but not to get them since this is a system-wide resource and may contain messages from multiple applications. As with the dead-letter queue (DLQ), it is customary that most applications have only `put` access to this queue.

Example 13-1 shows the **`setmqaut`** command that is required for user `mqcust1` on `QM1` to use the application queues and the AMS error queue. Be aware that the AMS error queue is used as a kind of dead-letter queue for signed or encrypted messages that do not reach their destination queue. Therefore, the application users must be allowed to put messages on that queue.

Example 13-1 setmqaut command run on QM1

```
setmqaut -m QM1 -n SYSTEM.PROTECTION.ERROR.QUEUE -t queue -g mqcustg1 -all +put
setmqaut -m QM1 -n SYSTEM.PROTECTION.POLICY.QUEUE -t queue -g mqcustg1 -all
+browse
```

Example 13-2 shows the **`setmqaut`** commands that are required on `QM2` for user `mqcust2` to use its application queues and the AMS error queue.

Example 13-2 setmqaut command run on QM2

```
setmqaut -m QM2 -n SYSTEM.PROTECTION.ERROR.QUEUE -t queue -g mqcustg2 -all +put
setmqaut -m QM2 -n SYSTEM.PROTECTION.POLICY.QUEUE -t queue -g mqcustg2 -all
+browse
```

13.2.3 Application configuration

The objective of this step is to generate keystores and keys for the applications.

The setup of the application user `mqcust1` keystore is similar to the one for the queue manager. The keystore is located in the hidden `.mqs` catalog in the user's home catalog. The differences between a queue manager's keystore and a private keystore is the name of the keystore and the name of the personal certificate. In an actual customer environment, the name of the keystore can be any name, as long as the extension is `.kdb`. Be aware that the label for this user also must start with `ibmwebspheremq` in lowercase characters. See 9.2.10, "Creating a key repository for users" on page 115 for a description of how to build the keystore.

In addition, for the application user `mqcust1`, a `keystore.conf` file must exist. This file is present in the user's home catalog. Example 13-3 shows the `keystore.conf` content.

Example 13-3 Contents of the keystore.conf file for mqcust1

```
cms.keystore = /home/mqcust1/.mqs/mqcust1key
cms.certificate = ibmwebspheremqmqcust1
```

13.3 Exchanging signed messages

The first stage of the WebSphere MQ AMS scenario uses signed messages only. The intention is to show a staged implementation of WebSphere MQ AMS functionality. The work that is needed to implement keystores and keys for signing messages is a lot simpler than the key management that must be in place for encryption. So doing signed messages as a first step might be an attractive approach to exploring AMS functions.

Now that the applications and queue manager have the necessary prerequisite configuration, all that is needed to initiate message signing is to define a policy of integrity against the application queues on `QM1` and `QM2`.

13.3.1 Integrity policy definition

To define security policies to messages in WebSphere MQ AMS, you can either use the WebSphere MQ Explorer or a command that typically can be scripted. In this scenario, the commands in Example 13-4 were used to define the policy on the two queue managers.

Example 13-4 setmqspl command issued for a signing policy on QM1 and QM2

```
On QM1:
setmqspl -m QM1 -p QA.Q2.AMS.S.QM2 -s SHA256 -e NONE -t 0

On QM2:
setmqspl -m QM2 -p Q2.AMS.S.QM2 -s SHA256 -e NONE -t 0
```

Where:

- `-m <qmgr name>`
- `-p <policy name>`
- `-s <signing algorithm>`
- `-e <encryption algorithm>`
- `-t <toleration 0=no | 1=yes>`

A short description of the naming standard used for the policy name:

- All alias queues start with a prefix of "QA".
- "Q2" is just a random queue name.
- "AMS" specifies that this is a queue with a security policy.
- "S" means that the policy is signing.
- "QM2" indicates that this queue resides at queue manager QM2. This alias points to a remote queue definition on QM1.

For a full description of the **setmqspl** command syntax, see the information center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/as30390.htm

13.3.2 Test sending and receiving signed messages

Example 13-5 shows mqcust1 on QM1 putting messages to a queue with the message signing policy.

Example 13-5 mqcust1 on QM1 and message signing policy

```
$ /opt/mqm/samp/bin/amqsputc QA.Q2.AMS.S.QM2 QM1
Sample AMQSPUT0 start
target queue is QA.Q2.AMS.S.QM2
Hello!! This message is signed

Sample AMQSPUT0 end
```

Example 13-6 shows mqcust2 on QM2 getting messages from a queue with the message signing policy.

Example 13-6 mqcust2 on QM2 and message signing policy

```
$ /opt/mqm/samp/bin/amqsgetc Q2.AMS.S.QM2 QM2
Sample AMQSGET0 start
message <Hello!! This message is signed>
no more messages
Sample AMQSGET0 end
```

Because WebSphere MQ enforces policy on the protected queue, it is difficult to retrieve a message off of the protected queue in its signed and encrypted state to verify that it has in fact been signed. However, if the sender channel from QM1 is stopped, the message may be inspected while on the transmission queue since that queue is not governed by a protection policy. Figure 13-2 on page 247 shows a hexadecimal dump of a signed message on the transmission queue.

The payload text is plainly visible but surrounded by WebSphere MQ AMS headers and trailers. The public key of the signer's X.509 certificate is part of the WebSphere MQ AMS message envelope and the certificate's issuer distinguished name (DN) is visible in the dump. The act of signing the message cryptographically binds the public key to the message such that any modification of the message envelope or its contents is detectable by the recipient and will cause the message to be diverted to the error queue.

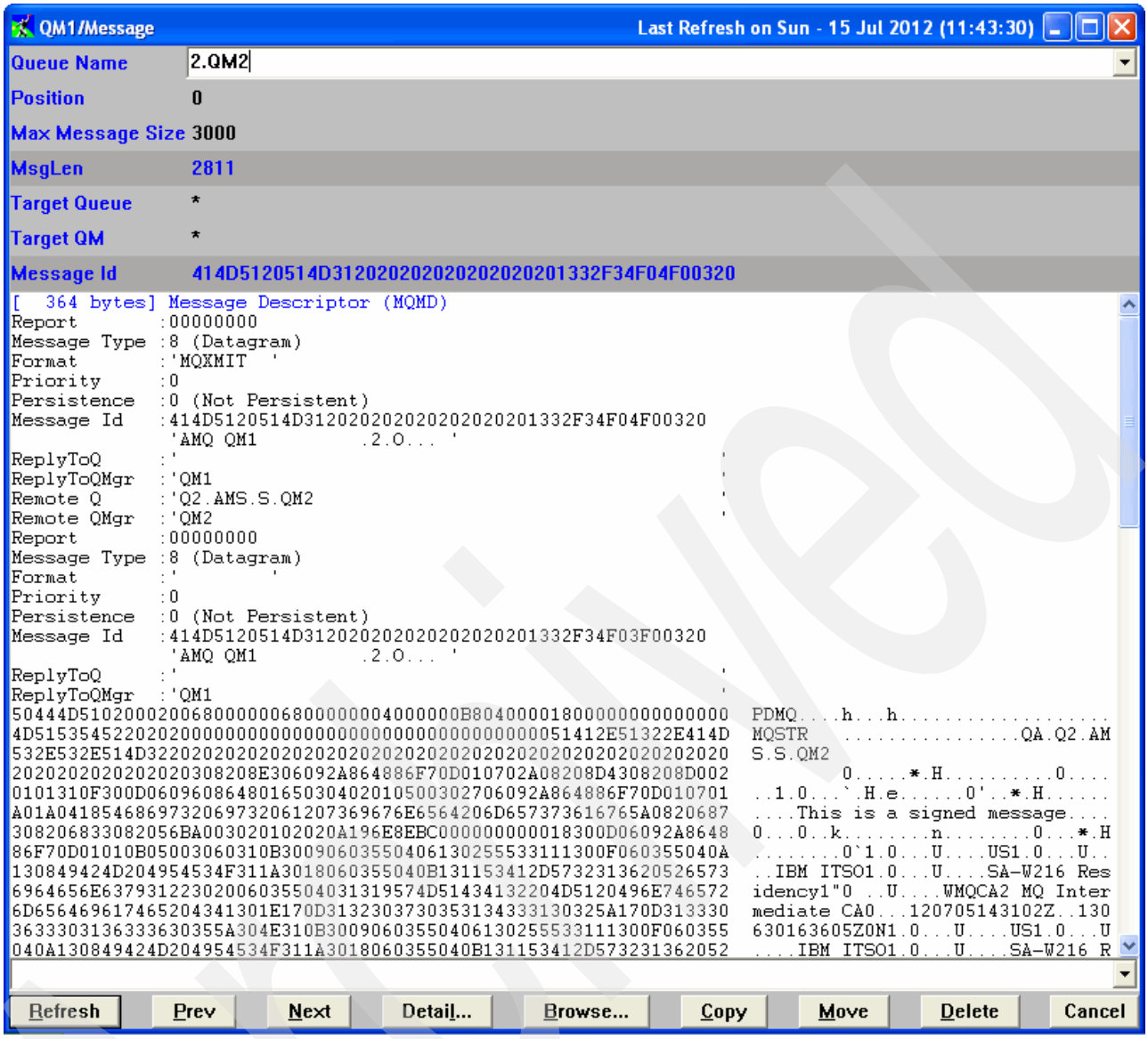


Figure 13-2 Signed message on QM1's transmission queue 2.QM2

MQMon program: The screen capture is of the MQMon program, which is freely downloadable as SupportPac MO71 from this website:

http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24000142

The index page for all SupportPacs is at this link:

http://www.ibm.com/software/integration/support/supportpacs

13.4 Exchanging encrypted messages

The previous step demonstrated the exchange of signed messages. In order to encrypt a message for a particular user with public/private key encryption, the sender must have the recipient's public key available. This step of the scenario shows the exchange of public keys between the two applications, updating the protection policy to specify encryption and demonstration with the new settings.

13.4.1 Key exchange

The act of creating a certificate generates both the public and private keys. The personal certificate always contains both of these keys; however, the public key can be extracted to a stand-alone file that contains the certificate identity, the identity of the CA signer, and the public key. These components are bound together cryptographically so that any modification of the file is detectable. The result is that the file containing the public key can be distributed over a non-secure channel, such as email or File Transfer Protocol (FTP).

Because the sender must have the recipient's public key in order to encrypt messages, an exchange of keys is required between participating messaging partners. In this example, encrypted messages flow from Customer 1 to Customer 2, therefore, the `mqcust1` account must have a local copy of the `mqcust2` public key. This entails Customer 2 extracting their public key and delivering that file by any convenient means to Customer 1.

Example 13-7 shows how to extract Customer 2's public key.

Example 13-7 How to extract a public key from mqcust2

```
Use the -extract attribute to create a public key for user mqcust2:
runmqakm -fips -cert -extract -db mqcust2key.kdb -stashed -label
ibmwebspheremqmqcust2 -target mqcust2.pub
```

The file containing the public key is sent to user `mqcust1` and stored in the default directory that is used by WebSphere MQ AMS. Since the scenario uses a Linux system, that directory location is `$HOME/.mqs`.

Next, the public key is added into the keystore of Customer 1 by using the command that is shown in Example 13-8.

Example 13-8 How to add a public key to mqcust1

```
To add the public key for user mqcust2 in the user mqcust1s keystore, do:

$ runmqakm -fips -cert -add -db mqcust1key.kdb -stashed -file mqcust2.pub -label
ibmwebspheremqmqcust2 -trust enable

Now, list the mqcust1 users keystore to check that it is right:
$ runmqakm -fips -cert -list -db mqcust1key.kdb -stashed
Certificates found
* default, - personal, ! trusted
!       WMQCA1 MQ Root CA
!       MQ Intermediate CA.cer
!       ibmwebspheremqmqcust2
-       ibmwebspheremqmqcust1
```

13.4.2 Encryption policy definition

There is a small change in the name of the queue that is used for encryption policy. The alias queue name is `QA.Q2.AMS.C.QM2`. The "`C`" in the name indicates an encrypted policy for this queue. The rest of the queue name has the same meaning as it does in the signing policy example. Also, this alias queue points to a remote queue definition on `QM1`.

Figure 13-3 shows the properties of an encryption policy on `QM1` from WebSphere MQ Explorer.

Note that when using the WebSphere MQ Explorer to add security policies to queues, it will not show cluster queues that are published from other queue managers. See Example 13-11 on page 252 for the command-line syntax and an example.

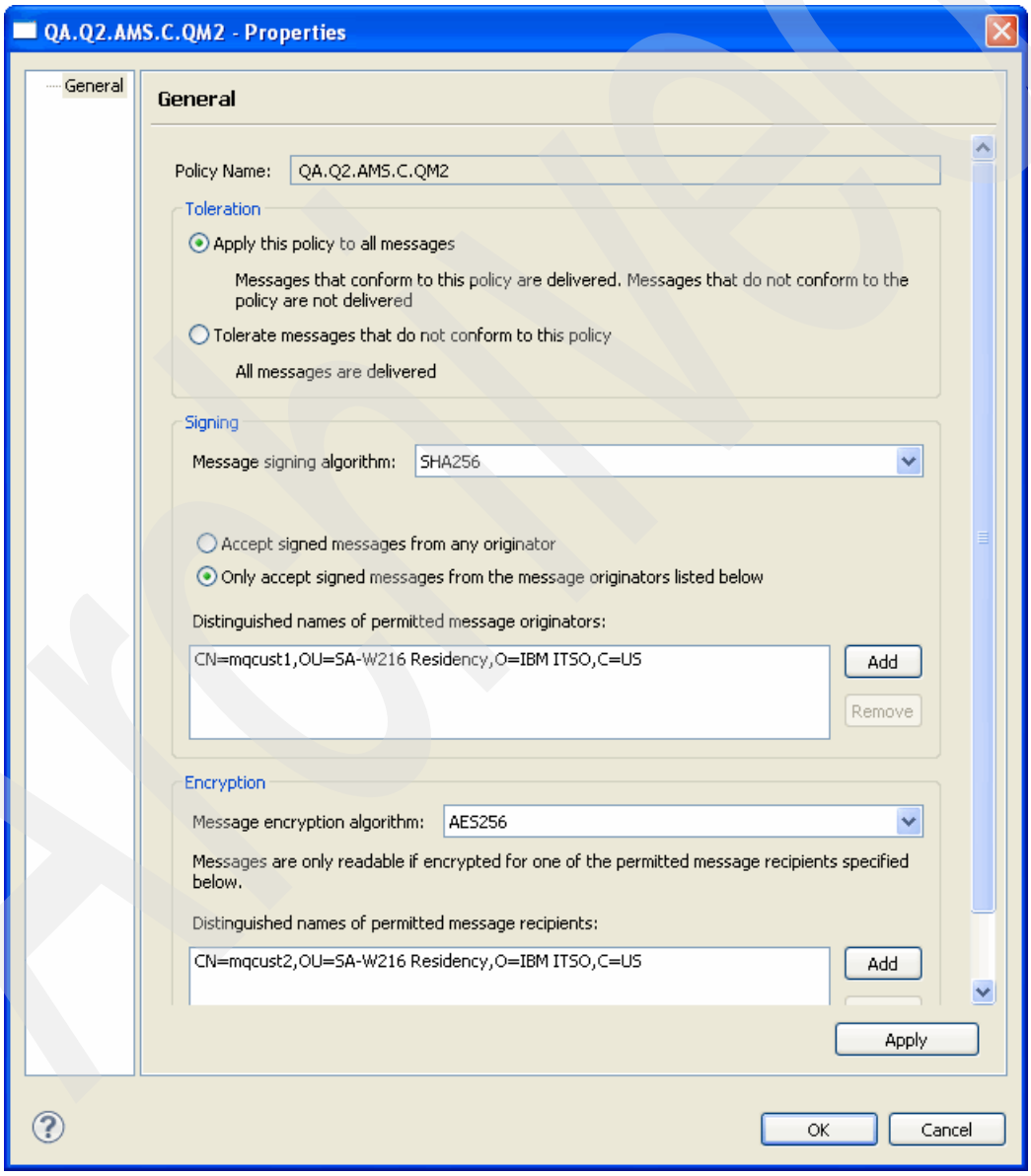


Figure 13-3 Encryption policy for alias queue QA.Q2.AMS.C.QM2 on QM1

13.4.3 Testing the send and receive of encrypted messages

Example 13-9 shows `mqcust1` on `QM1` putting messages to a queue with message encryption policy.

Example 13-9 mqcust1 on QM1 putting encrypted message

```
$ /opt/mqm/samp/bin/amqsputc QA.Q2.AMS.C.QM2 QM1
Sample AMQSPUT0 start
target queue is QA.Q2.AMS.C.QM2
Hello world!! This message is encrypted

Sample AMQSPUT0 end
```

Example 13-10 shows `mqcust2` on `QM2` getting messages from a queue with message encryption policy.

Example 13-10 mqcust2 on QM2 getting encrypted message

```
$ /opt/mqm/samp/bin/amqsgetc Q2.AMS.C.QM2 QM2
Sample AMQSGET0 start
message <Hello world!! This message is encrypted>
no more messages
Sample AMQSGET0 end
```

Important: As a user of WebSphere MQ Advanced Message Security, it is not possible to see that the message has been signed or signed and encrypted on its way through the MQ network. This is completely transparent to the application.

When trapping the message on a transmission queue, it is easy to see that the message has a different format and has grown in size. Figure 13-4 on page 251 shows a hexadecimal dump of the encrypted message on QM1's transmission queue `2.QM2`.

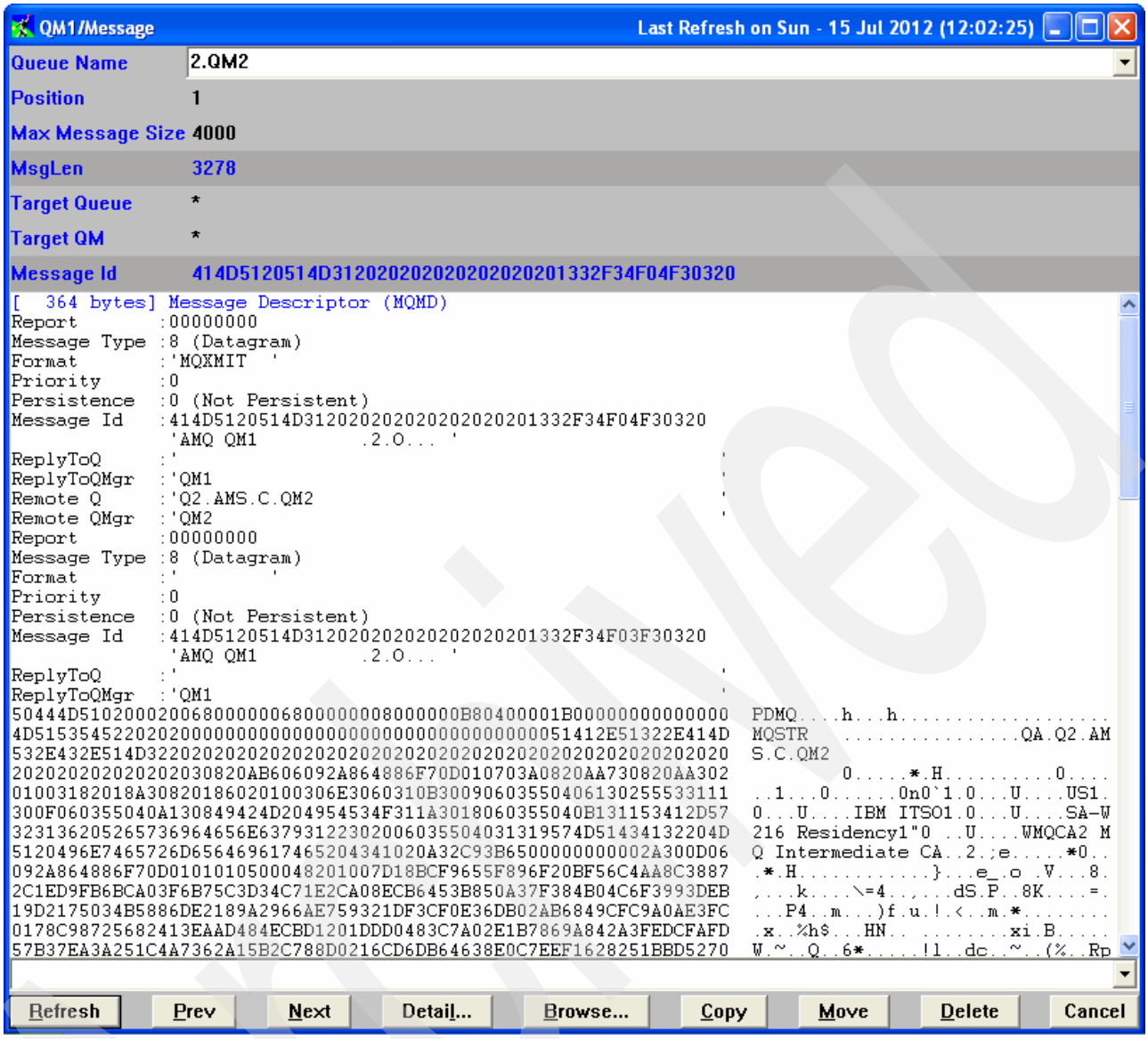


Figure 13-4 Encrypted message on QM1's transmission queue 2.QM2

Now, it is impossible to see the payload of the message as it is encrypted.

13.5 Summary

The point-to-point application scenario used WebSphere MQ Advanced Message Security to implement secure messaging. This demonstrated methods to provide the integrity and privacy of message data.

13.6 Further considerations

Be aware that when defining security policies to cluster queues that are published from remote queue managers, the cluster queues will not appear in the WebSphere MQ Explorer.

There are two possible ways of defining such policies, in the WebSphere MQ Explorer panel by specifying the policy name yourself, or by using the command interface. Example 13-11 shows how this can be done using the **setmqspl** command.

Example 13-11 setmqspl use for cluster queues or scripts

```
setmqspl -m QM1 -p SOME.CLUSTER.Q -s SHA256 -e AES256 -r "CN=mqcust2,OU=SA-W216
Residency,O=IBM ITSO,C=US" -t O
```

Figure 13-5 shows the result in the WebSphere MQ Explorer.

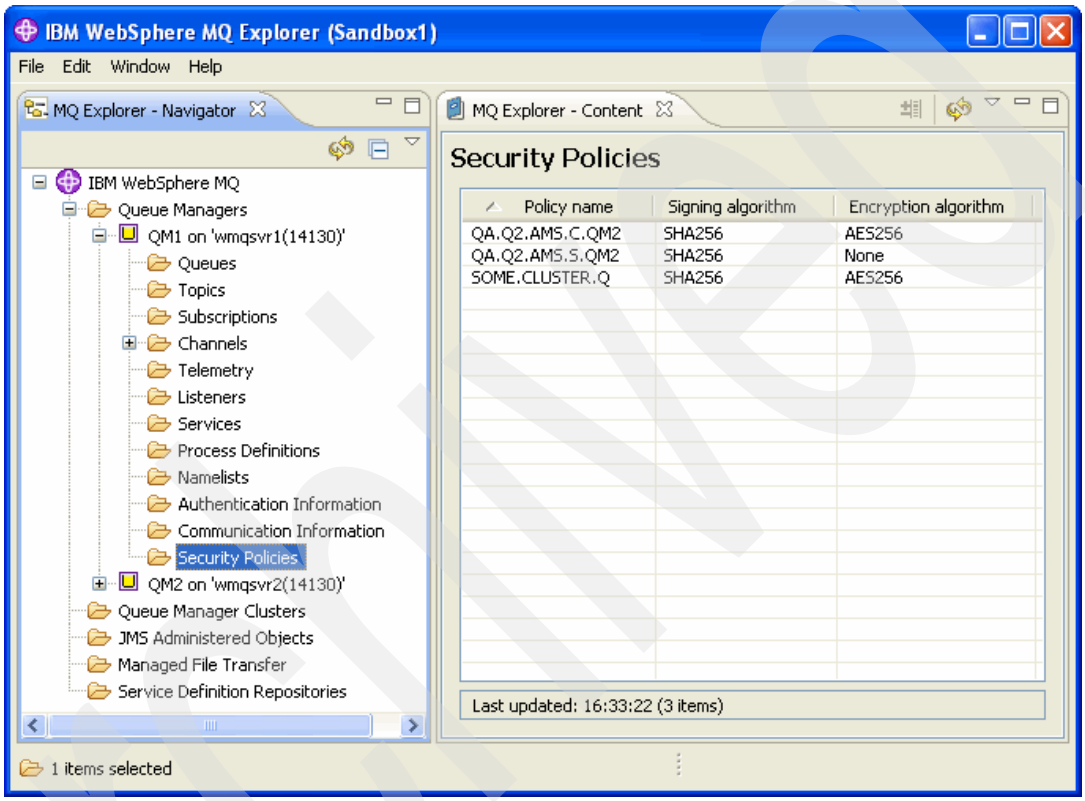


Figure 13-5 AMS security policies on QM after setmqspl added a cluster queue

14

Scenario: WebSphere MQ AMS revocation checking

Certificate revocation checking is a new feature in the IBM WebSphere MQ Advanced Message Security (AMS) 7.0.1.1 Fix Pack. When a certificate is revoked prior to its expiration date, the certificate authority (CA) publishes the revocation to a certificate revocation list (CRL) or Online Certificate Status Protocol (OCSP) responder. Revocation checking is the means by which a program using WebSphere MQ AMS can obtain the current status of certificates associated with messages. Without revocation checking, certificates are considered valid until their expiry, regardless of their status at the CA.

This chapter presents a scenario showing how to configure revocation checking in WebSphere MQ AMS. The scenario demonstrates applications putting and getting messages to and from queues by using a WebSphere MQ client, and describes how to use OCSP and CRL in this configuration. Base WebSphere MQ uses only Lightweight Directory Access Protocol (LDAP) for CRL lookup, but in addition to LDAP, WebSphere MQ AMS can use CRL files, and the scenario explains how this is accomplished as well.

This chapter contains the following topics:

- Scenario overview
- Implementing the scenario
- Testing the certificate revocation
- Summary
- Further considerations

14.1 Scenario overview

With WebSphere MQ AMS, you can verify the validity of a certificate by using either OCSP or CRL. WebSphere MQ AMS can be configured for either OCSP checking, CRL checking, or both. If both methods are enabled, OCSP is used first. If the revocation status of a certificate is unknown after the OCSP check, WebSphere MQ AMS uses CRL checking.

For a more comprehensive discussion of revocation checking, CRL, and OCSP, see Chapter 12, “Scenario: CRL/OCSP certificate revocation” on page 219.

Important: Messages signed or encrypted by using a certificate that has been revoked will not be processed by WebSphere MQ AMS when revocation checking is enabled. This includes cases where the messages signed by a revoked certificate were produced before the revocation was published or if revocation is enabled on the consumer side but not the producer side of the interface. In the event that it is necessary to process messages signed or encrypted with revoked certificates, revocation checking must be disabled on the consumer side.

For additional advice about handling certificates before they are revoked, see 14.5, “Further considerations” on page 262.

14.1.1 Scenario design

The scenario uses two endpoint applications that utilize WebSphere MQ AMS for signing or encrypting messages. The two applications connect as WebSphere MQ clients to separate queue managers. The LDAP and OCSP responders and their CRLs are located on one or more LDAP servers. This is the same basic setup as used in Chapter 13, “Scenario: End-to-end security using WebSphere MQ AMS” on page 241.

For this scenario, we will use one LDAP server and one OCSP responder to illustrate how it works. The WebSphere MQ application will not run if revocation checking is enabled and the revocation responders are not available. For this reason, the LDAP and OCSP responders in a production environment are usually made highly available.

The queue managers can be linked by distributed channels, a WebSphere MQ cluster, intermediate queue managers, or any other valid WebSphere MQ architecture; it does not affect the WebSphere MQ AMS configuration. Of course, all channels in this scenario are Secure Sockets Layer (SSL)-enabled.

The applications in this scenario connect to WebSphere MQ using client channels. This means that the client application certificates do not exist on the queue manager system, thus preventing a WebSphere MQ administrator from being able to see or tamper with the messages. Only the applications that have a valid certificate have access to the messages. In most cases, this situation would be regarded as beneficial, but it could limit the administrator in debugging efforts.

This scenario will address the following areas of security with regard to WebSphere MQ AMS:

- Use of LDAP CRLs
- Use of OCSP responders

Some topics are not addressed in this scenario because it is assumed that you are already familiar with these topics:

- Setup of queue managers
- Definition of queues
- Definition of SSL channels
- Basic protection of queue managers, queues, channels, namelists, and file system
- Definition of WebSphere MQ AMS security policies

The diagram in Figure 14-1 shows the applications and queue managers in the revocation checking scenario.

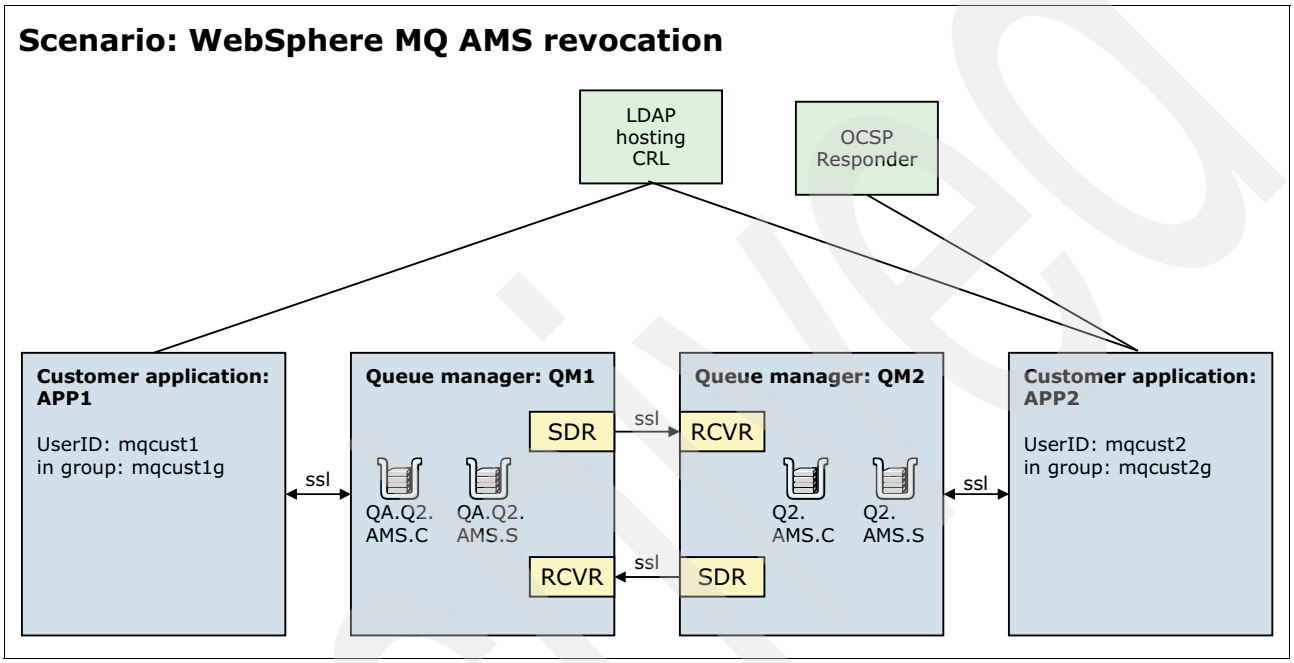


Figure 14-1 WebSphere MQ AMS revocation checking scenario

In this scenario, there are two queue managers and two client applications. This makes the scenario somewhat more complex and hopefully more in line with the actual production topologies.

The scenario uses one-way communication from APP1 to APP2 to simplify the setup and test environment. The intent of the scenario is to show revocation checking and not a complex configuration. The scenario does not implement revocation checking on the queue manager level, only for the client applications, APP1 and APP2.

There is no measurement of performance impact in such a configuration. Just be aware that encrypting messages requires more resources.

Finally, with the end-to-end security topology as shown in this scenario, the WebSphere MQ administrator is prevented from reading message payloads that flow between the client applications.

For more information about the systems and infrastructure that are used in this topology, see Chapter 8, “Scenario preparation” on page 83.

14.1.2 Prerequisites

This scenario requires that WebSphere MQ Server with AMS is installed on the servers. In addition, WebSphere MQ Client version 7.5 with the included AMS functionality must be installed and available on both the client application systems. The clients each have an application installed (APP1 and APP2).

An LDAP server is available and reachable from the participating queue managers and the involved client application servers. In addition, OCSP responders must be available for all participating partners.

WebSphere MQ queue managers and their infrastructure with established communications are in place. SSL is configured for all channels that will be used in this scenario. Finally, certificate keystores must be configured and available at all four places.

14.2 Implementing the scenario

The following sections describe how to configure the clients for revocation checking. The client with APP1 and the `mqcust1` user will use LDAP only. The second client with APP2 and the `mqcust2` user will use both OCSP and LDAP.

14.2.1 Queue manager configuration

The setup for the WebSphere MQ AMS revocation checking scenario uses default values whenever possible. The intention is to show how it works, not to tune for a specific environment. It is assumed that the applications and queue managers are already exchanging messages so the only task in WebSphere MQ AMS V7.5 is to provision keystores and keys for the applications.

14.2.2 Client configuration

The keystore configuration file for the client applications must be updated with information about the LDAP server or the OCSP responder (or both) to be able to perform revocation checks.

> **Important:** The `keystore.conf` file owner must ensure that only the file owner is entitled to read the file.

Example 14-1 shows how this keystore configuration is done for LDAP. Both client users `mqcust1` and `mqcust2` have these entries in their `keystore.conf` files.

Example 14-1 LDAP information in keystore.conf

```
crl.ldap.host=wmqldap1
crl.ldap.port=389
crl.cdp=off
crl.ldap.version=3
crl.ldap.user=cn=mqmbind,ou=bindusers
crl.ldap.pass=<password>
```

Example 14-2 on page 257 shows the OCSP information in the `keystore.conf` file. Only client user `mqcust2` has these entries in the `keystore.conf` file.

Example 14-2 OCSP information in keystore.conf

```
ocsp.enable=on
ocsp.url='http://wmqca2.saw216.itso.ibm.com/ocsp'
```

To ensure that the client will perform CRL checking, the `mqclient.ini` should be configured as shown in Example 14-3. Only the `mqcust2` user has this information in the `mqclient.ini` file

Example 14-3 mqclient.ini file with OCSP

```
SSL:
   OCSPCheckExtensions=YES
   SSLFipsRequired=YES
```

14.2.3 Applications

The amqsputc and amqsgetc sample programs are used to verify the WebSphere MQ AMS configuration. They are run in client mode from the application servers. The intent is to show that applications on servers with revoked certificates cannot get the messages. These applications are written in C.

14.3 Testing the certificate revocation

Three scenarios are tested, in the following order:

1. Revoking the receiver's certificate (`mqcust2`)
2. Revoking the sender's certificate (`mqcust1`)
3. Renewing both certificates

14.3.1 Test 1: Revoking the receiver's certificate

In the first test, messages are flowing from APP1 to APP2. Then, the certificate for `mqcust2` (the user ID for APP2) is revoked and that revocation is recognized by the AMS component associated with APP2 using OCSP. APP1 is able to continue sending messages, but APP2 cannot receive them.

To verify that APP1 can put new messages on the queues, use any tool that can display the depth of a queue. The queue depth should increase.

Example 14-4 shows what happens when APP2, using the `mqcust2` user ID, tries to get a message from a queue after the certificate is revoked.

Example 14-4 mqcust2 user with revoked certificate

```
$ /opt/mqm/samp/bin/amqsgetc Q2.AMS.S.QM2 QM2
Sample AMQSGET0 start
MQCONN ended with reason code 2393
```

Example 14-5 shows the client's error log on QM2 at the same time.

Example 14-5 The /var/mqm/errors/AMQERR01.LOG on QM2

```
07/15/2012 04:02:24 PM - Process(29746.1) User(mqcust2) Program(amqsbcgc)
```

```
                    Host(wmqsvr2.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(.)

AMQ9665: SSL connection closed by remote end of channel 'APP2.SVRCONN'.

EXPLANATION:
The SSL or TLS connection was closed by the remote host 'wmqsvr2
(192.168.102.102)(14130)' during the secure socket handshake. The channel is
'APP2.SVRCONN'; in some cases its name cannot be determined and so is shown as
'????'. The channel did not start.
ACTION:
Check the remote end of the channel for SSL and TLS errors. Fix them and
restart the channel.
```

Example 14-6 shows the queue manager's error log on QM2.

Example 14-6 The /var/mqm/qmgrs/QM2/errors/AMQERR01.LOG

```
07/15/2012 04:02:24 PM - Process(23345.318) User(mqm) Program(amqrmppa)
                    Host(wmqsvr2.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(QM2)

AMQ9633: Bad SSL certificate for channel '????'.

EXPLANATION:
A certificate encountered during SSL handshaking is regarded as bad for one of
the following reasons:
(a) it was formatted incorrectly and could not be validated
(b) it was formatted correctly but failed validation against the Certification
  Authority (CA) root and other certificates held on the local system
(c) it was found in a Certification Revocation List (CRL) on an LDAP server
(d) a CRL was specified but the CRL could not be found on the LDAP server
(e) an OCSP responder has indicated that it is revoked

The channel is '????'; in some cases its name cannot be determined and so is
shown as '????'. The remote host is 'wmqsvr2 (192.168.102.102)'. The channel
did not start.

The details of the certificate which could not be validated are
'[Class=]GSKVALMethod::X509[Issuer=]CN=WMQCA2 MQ Intermediate CA,OU=SA-W216
Residency,O=IBM
ITSO,C=US[#=]32c93b6500000000002a[Subject=]CN=mqcust2,OU=SA-W216
Residency,O=IBM ITSO,C=US[Class=]GSKVALMethod::X509[Issuer=]CN=WMQCA2 MQ
Intermediate CA,OU=SA-W216'.

The certificate validation error was 575032.
ACTION:
Check which of the possible causes applies on your system. Correct the error,
and restart the channel.
----- amqccisa.c : 6717 -------------------------------------------------------
07/15/2012 04:02:24 PM - Process(23345.318) User(mqm) Program(amqrmppa)
                    Host(wmqsvr2.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(QM2)

AMQ9492: The TCP/IP responder program encountered an error.
```

```
EXPLANATION:
The responder program was started but detected an error.

The host name was 'wmqsvr2 (192.168.102.102)'; in some cases the host name
cannot be determined and so is shown as '????'.
ACTION:
Look at previous error messages in the error files to determine the error
encountered by the responder program.
```

The queue manager's error log clearly indicates that there is something wrong about the certificate validation. The reported error is `575032`, which means the certificate is revoked.

For a complete list of SSL and Transport Layer Security (TLS) return codes, see the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/topic/com.ibm.mq.doc/fm18810_.htm

The number of messages on the queue remains the same, which means that these messages are stuck on the queue.

14.3.2 Test 2: Revoking the senders certificate

In the second test, messages are flowing from APP1 to APP2, but APP2 is not able to receive them. Then, the certificate for `mqcust1` (the user ID for APP1) is also revoked and that revocation is recognized by the AMS component associated with APP1 using LDAP and CRL. APP1 is no longer able to send messages.

The error messages in the logs will look similar to the messages received in the previous test.

Example 14-7 shows the client error message when the certificate is revoked.

Example 14-7 mqcust1 trying to use queue when certificate is revoked

```
$ /opt/mqm/samp/bin/amqsputc QA.Q2.AMS.S.QM2 QM1
Sample AMQSPUT0 start
MQCONN ended with reason code 2393
```

Example 14-8 shows that the queue manager error log for QM1 has a failure similar to that experienced by APP2 for the `mqcust2` user in the previous test.

Example 14-8 The /var/mqm/qmgrs/QM1/errors/AMQERR01.LOG

```
07/16/2012 10:43:21 AM - Process(10778.182) User(mqm) Program(amqrmppa)
                    Host(wmqsvr1.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(QM1)

AMQ9633: Bad SSL certificate for channel '????'.

EXPLANATION:
A certificate encountered during SSL handshaking is regarded as bad for one of
the following reasons:
(a) it was formatted incorrectly and could not be validated
(b) it was formatted correctly but failed validation against the Certification
  Authority (CA) root and other certificates held on the local system
(c) it was found in a Certification Revocation List (CRL) on an LDAP server
(d) a CRL was specified but the CRL could not be found on the LDAP server
```

```
(e) an OCSP responder has indicated that it is revoked

The channel is '????'; in some cases its name cannot be determined and so is
shown as '????'. The remote host is 'wmqsvr1 (192.168.102.101)'. The channel
did not start.

The details of the certificate which could not be validated are
'[Class=]GSKVALMethod::X509[Issuer=]CN=WMQCA2 MQ Intermediate CA,OU=SA-W216
Residency,O=IBM
ITSO,C=US[#=]196e8ebc000000000018[Subject=]CN=mqcust1,OU=SA-W216
Residency,O=IBM ITSO,C=US[Class=]GSKVALMethod::X509[Issuer=]CN=WMQCA2 MQ
Intermediate CA,OU=SA-W216'.

The certificate validation error was 575032.
ACTION:
Check which of the possible causes applies on your system. Correct the error,
and restart the channel.
```

And, the number of messages on the queue remain the same. The sender application is not able to put new messages.

14.3.3 Test 3: Both certificates are renewed

In the third test, the revoked certificates for `mqcust1` and `mqcust2` are renewed. APP1 can now put messages to the queue. APP2 can now receive new messages. However, APP2 cannot receive messages that were sent by APP1 while the certificate for `mqcust2` was revoked.

This situation can occur if a certificate expires and needs to be reactivated. For certificates that are revoked deliberately, the certificate would not normally be renewed. Be aware that in this situation, the unprocessed messages intended for the revoked receiver should be handled according to business and security guidelines.

Example 14-9 shows how to request a renewal for a certificate. The advantage is that it keeps the label and the distinguished name (DN). Remember to do the same for `mqcust2`.

Example 14-9 mqcust1 renew certificate request

```
runmqakm -fips -certreq -recreate -db mqcust1key.kdb -stashed -label
ibmwebspheremqmqcust1 -target keyv2.req -sigalg sha256
```

The renew request must be handled in the same way as the original certificate request, and the signed certificate must be received in the keystore (see 8.11, “WebSphere MQ (CMS) keystores” on page 92).

Sending messages with the renewed certificate

Now that the renewed certificate for `mqcust1` is in the keystore for APP1, using `mqcust1` to put messages to a queue with a signed message policy defined works fine.

However, the public key for `mqcust2`’s renewed certificate has not been received into the keystore for APP1. When APP1 attempts to send messages that should be encrypted, the send fails because the public certificate of user `mqcust2` is still the revoked version in `mqcust1`’s keystore.

Attempting to receive previously sent messages with the renewed certificate

When APP2 (as `mqcust2`) attempts to receive encrypted messages that were left on the queue after the certificate for `mqcust2` was revoked, the attempt fails because the messages were produced by a user (`mqcust1`) who was using the public key of a user (`mqcust2`) with a revoked certificate.

The failing messages go to the SYSTEM.PROTECTION.ERROR.QUEUE with the mqrc of `2063 Security Error`.

Example 14-10 shows the `/var/mqm/errors/AMQERR01.LOG` when `mqcust2` attempts to receive a message that was produced with a revoked certificate.

Example 14-10 mqcust2 receiving message with revoked certificate

```
07/17/2012 09:18:20 AM - Process(4945.1) User(mqcust2) Program(amqsgetc)
                    Host(wmqsvr2.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(.)

AMQ9017: WebSphere MQ security policy internal error: message could not be
unprotected: GSKit error code 851968, reason 43.

EXPLANATION:
The WebSphere MQ security policy interceptor could not verify or decrypt a
message because the indicated GSKit error occurred. This can happen for several
reasons, all of which are internal failures: (1) the message is not a valid
PKCS#7 message; (2) the sender's certificate does not have the required key
usage bit to be able to encrypt the message; (3) the sender's certificate was
not recognized as a trusted certificate; (4) receiver is not among the
recipients of the message.
ACTION:
Consult the GSKit information in the Information Center for the explanation of
the GSKit reason code and take corrective action. If the problem persists,
contact your IBM service representative.
----- smqodida.c : 1655 -------------------------------------------------------
07/17/2012 09:18:20 AM - Process(4945.1) User(mqcust2) Program(amqsgetc)
                    Host(wmqsvr2.saw216.ibm.com) Installation(Installation1)
                    VRMF(7.5.0.0) QMgr(.)

AMQ9044: The WebSphere MQ security policy interceptor has put a defective
message on error handling queue SYSTEM.PROTECTION.ERROR.QUEUE.

EXPLANATION:
This is an informational message that indicates the WebSphere MQ security
policy put a message it could not interpret on the specified error handling
queue.
ACTION:
Make sure only valid messages are put onto queues protected by WebSphere MQ
security policies.
```

Exchange public certificates

Now that the certificates have been renewed, the public keys must be exchanged. Use the **`runmqakm`** command to extract the public certificates of the `mqcust1` and `mqcust2` users, and add the certificates at the opposite keystore.

Retry sending with renewed certificate

When sending with the renewed certificates and the renewed public certificate of the opposite party in place, both signed and encrypted messages can now be sent.

Retry receiving with renewed certificate

When receiving with the renewed certificates and the renewed public certificate of the opposite party in place, the receive fails for the old messages that had the revoked user certificate. The failing messages go to the SYSTEM.PROTECTION.ERROR.QUEUE with the reason code `2063`.

Receiving messages that were created with the renewed certificates works for both signed and encrypted messages.

14.4 Summary

The use of certificate revocation lists works for both OCSP responders and LDAP CRLs. Configuring the use of either of these two configurations is very simple in WebSphere MQ Advanced Message Security. The samples here have not considered the setup or use of any specific CRL provider, but just demonstrated that it works.

The important thing to be aware of is the fact that messages produced by a user with a revoked certificate will be unavailable from the queue. They will not block the queue as they will go to the SYSTEM.PROTECTION.ERROR.QUEUE with a mqrc `2063 - Security Error`, when someone tries to get them, but they can be regarded as lost for the receiving application.

14.5 Further considerations

For installations whose certificates have a short lifetime and who perform planned maintenance with replacing certificates, it is important to re-create and receive the new certificate before the old certificate expires.

Installing the renewed certificate before it expires will allow the messages produced or consumed with the old certificate to still be valid. If a certificate is revoked before it is renewed, there is a risk of losing those messages that have been produced with the revoked certificate, and then the renewal process will not solve the problem with the unprocessed messages on the receiver side. These messages will for all means be lost, go to the error queue, or be unavailable.

A

Working with the itsoME message exit

This appendix contains a message exit that is used to ease integration in a business-to-business (B2B) world. The usage of the exit is described further in Chapter 10, “Scenario: Securing IBM WebSphere MQ connections to connect a business partner” on page 141.

The message exit can also be used for internal communications to help applications adapt to changes in the network. In this case, the itsoME message exit might be able to help you, or inspire you to develop your own.

This appendix contains the following topics:

- Description
- Inbound traffic
- Outbound traffic
- Configuring the exit
- Compiling the exit
- Installing the exit
- Design considerations
- Debugging
- Message exit source

A.1 Description

The itsoME exit is used to protect the business from threats that are posed by messages coming from the outside world. The potential attacks that it can defend against include malicious redirection of messages using created ReplyTo headers or the dead-letter queue handler.

The exit also provides a mechanism to map routes, both inbound and outbound in order to provide a measure of isolation between the internal and external network namespaces. This prevents name collisions between the internal and external networks. It also limits the ability of external partners to enumerate internal resources.

For further information about how you can use this exit, see Chapter 10, “Scenario: Securing IBM WebSphere MQ connections to connect a business partner” on page 141.

A.2 Inbound traffic

Figure A-1 shows how itsoME might be used to mitigate threats for messages that are inbound from external third parties.

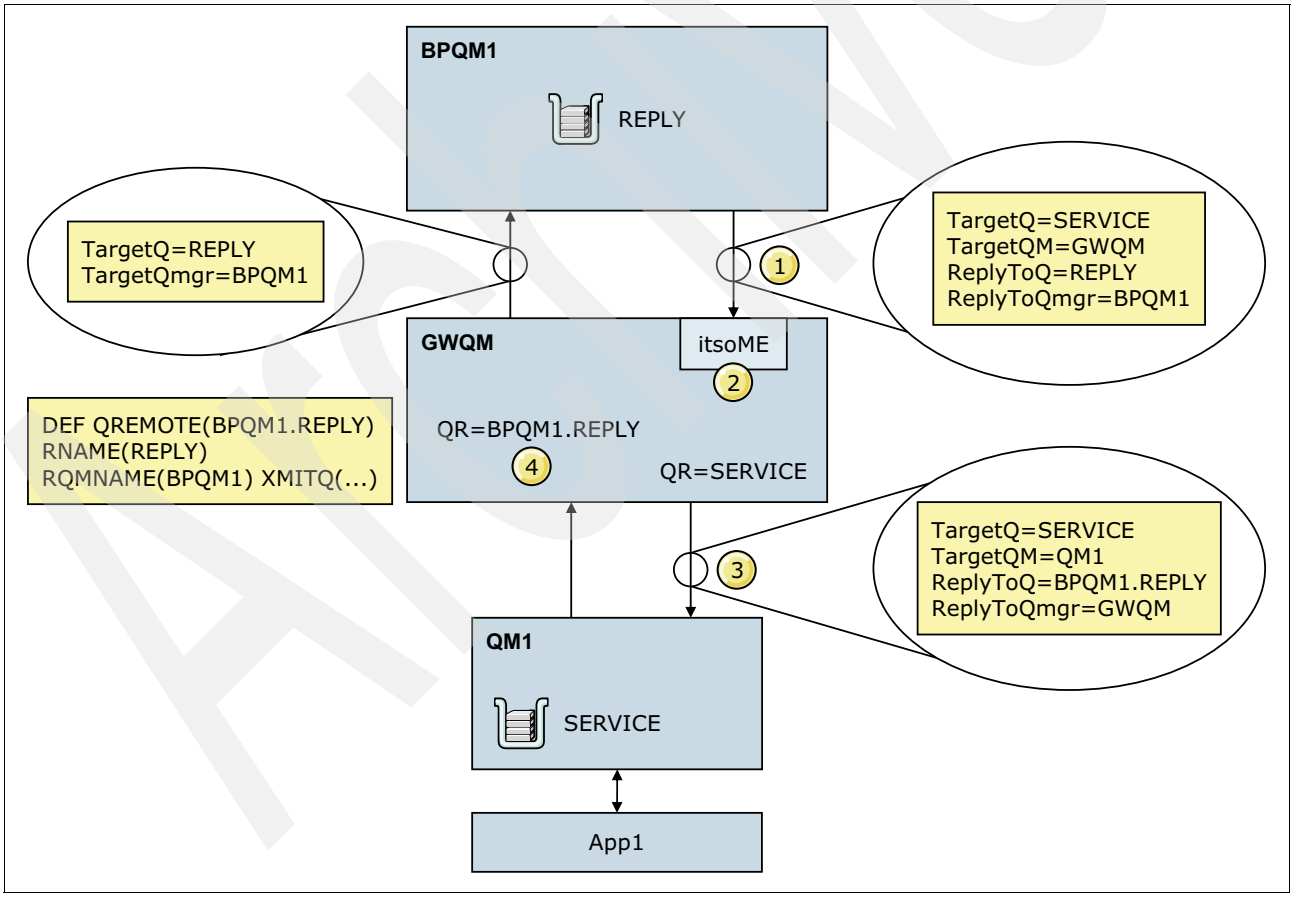


Figure A-1 How to use itsoME on inbound traffic, RTMQ=GWQM/

In Figure A-1, a message is received from a business partner's queue manager BPQM1 on the internal gateway queue manager GWMQ. The message is destined for a queue named SERVICE on the internal queue manager QM1. The requester wants the reply delivered to the REPLY queue on BPQM1.

The following steps in the message flow are shown in Figure A-1 on page 264:

1. The received message contains information about where the reply should be sent.
2. Message exit itsoME changes the ReplyToQmgr=GWMQ, and prefixes the ReplyToQ with the original ReplyToQmgr (BPQM1).
3. The message is sent to QM1 for processing.
4. APP1 sends a reply back to GWMQ for TargetQ=BPQM1.REPLY. This is resolved in GWQM to TargetQ=REPLY on BPQM1 by the QREMOTE definition.

A.2.1 Message exit functions

This section describes the various functions that are performed by the message exit.

Route mapping

The exit maps arbitrary external destinations to names that are controlled and resolved internally. In the example, the REPLY queue on the external business partner's queue manager BPQM1 is made to resolve to a destination on the gateway queue manager. The benefit is that none of the names used by the external business partner will conflict with internal names. If there are multiple business partners, this also prevents collision when multiple business partners happen to use identical names for their queues and queue managers. The namespace isolation also limits the ability of third parties to enumerate names on the internal network.

User Identifier mapping

When messages from multiple sources land on the same queue, the value of the User Identifier in the message descriptor can identify the sender of the message. But that field can be trusted only when the message originates on a queue manager that is secured and controlled by the internal administration team. Messages that arrive from external sources may contain any arbitrary value in the User Identifier field, including values that internally resolve to administrative accounts.

The itsoME exit addresses this issue by mapping the channel's MCAUSER value to the User Identifier field in the message descriptor of all inbound messages. If the security controls illustrated in the B2B scenario have been followed, an account representing the business partner has been created, authorized to a restricted subset of queues, and configured in the channel's MCAUSER. The exit takes advantage of the presence of this user ID by mapping it to the User Identifier field in all inbound messages. Provided that unique user IDs have been defined for each business partner, this method allows messages landing on a common request queue to be reliably traced back to their point of origin.

Report option cleansing

WebSphere MQ provides report options that allow the sender of a message to request notification at certain points in the message path. The reports are delivered as messages addressed to the destination specified in the ReplyTo fields of the sent message. One possible abuse of this functionality is to craft messages designed to generate reports addressed to queues to which the attacker does not have legitimate access.

The itsoME message exit addresses this issue by optionally clearing out the report options fields in the message descriptor of messages as they traverse the channel.

A.2.2 Configuring the queues

Example A-1 shows the commands to define the queue definitions for the business partner's BPQM1 queue manager.

Example A-1 Definitions in BPQM1

```
DEFINE QREMOTE(SERVICE) RNAME(SERVICE) RQMNAME(GWQM) XMITQ(GWQM)

DEFINE QLOCAL(GWQM) USAGE(XMITQ)

DEFINE QLOCAL(REPLY)
```

Example A-2 shows the commands to define the queue definitions for the internal gateway queue manager GWQM. The example also shows how the definition of the channel is altered to install the message exit.

Example A-2 Definitions in GWQM

```
DEFINE QREMOTE(BPQM1.REPLY) RNAME(REPLY) RQMNAME(BPQM1) XMITQ(BPQM1)

DEFINE QLOCAL(BPQM1) USAGE(XMITQ)

DEFINE QREMOTE(SERVICE) RNAME(SERVICE) RQMNAME(APP1) XMITQ(QM1)

DEFINE QLOCAL(QM1) USAGE(XMITQ)

ALTER CHANNEL(BPQM1.GWQM) CHLTYPE(RCVR) +
      MSGEXIT('itsoME(MsgExit)') +
      MSGDATA('RTQM=GWQM/')
```

Example A-3 shows the commands to define the queue definitions for queue manager QM1.

Example A-3 Definitions in QM1

```
DEFINE QLOCAL(SERVICE)

DEFINE QLOCAL(GWQM) USAGE(XMITQ)
```

A.3 Outbound traffic

When sending request messages to third parties, the itsoME message exit can control how reply messages are routed, without requiring changes to the applications. This provides the same benefits of isolating the network namespaces that are described in the previous section.

In the scenario that is shown in Figure A-2 on page 267, APP1, APP2, and APP3 are designed to receive their replies in a queue named REPLY. Each application is connected to a local instance of the REPLY queue and requires that reply messages be routed to the correct instance. Native WebSphere MQ name resolution requires that the four internal queue managers (GWQM, APP1, APP2, APP3) are known to BPQM1. Typically, this is achieved by defining queue manager alias definitions on the external queue manager BPQM1 for each internal queue manager.

The itsoME message exit addresses this issue by dynamically substituting the internal ReplyTo names with alternative names that resolve to objects defined on the gateway queue manager.

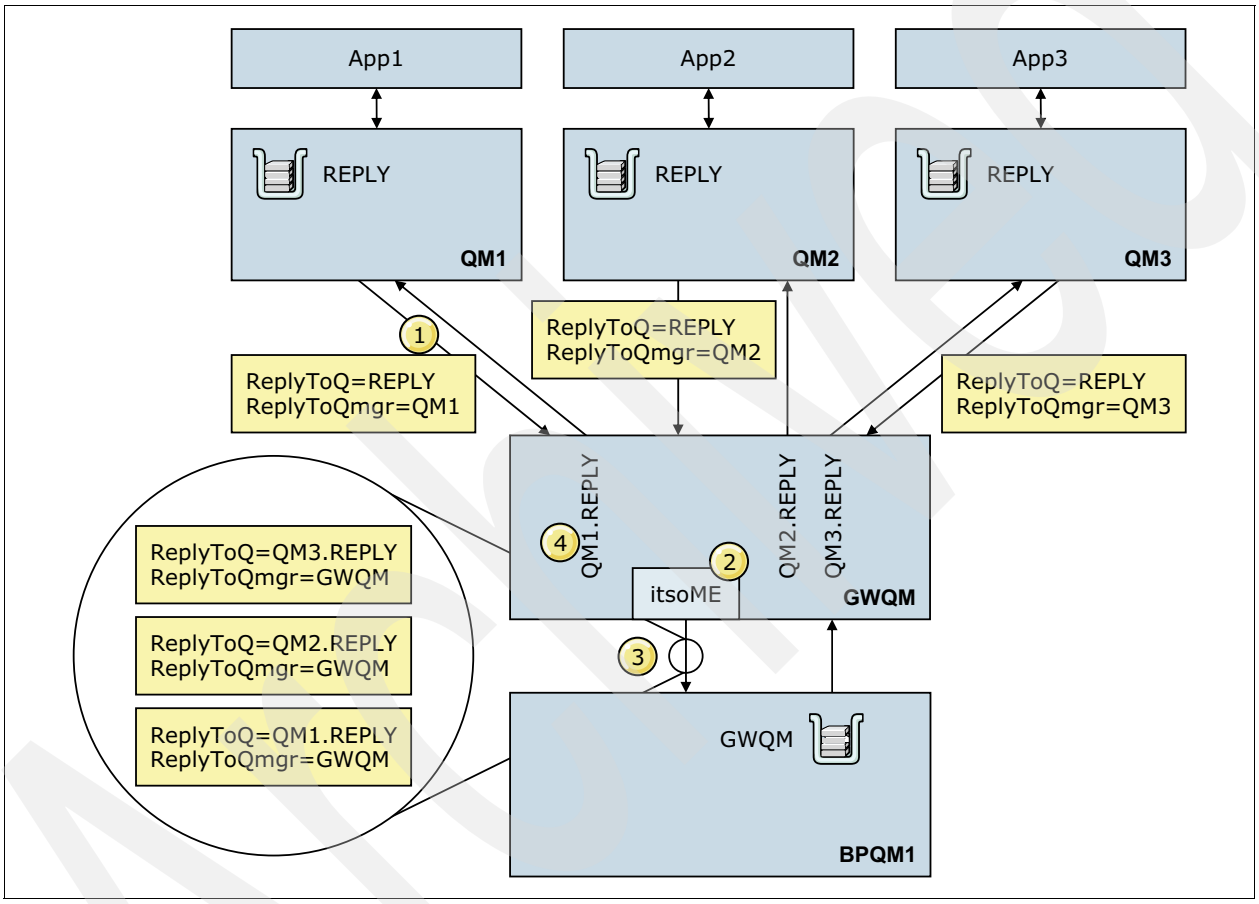


Figure A-2 How to use itsoME on outbound traffic, RTMQ=GWQM/

Using message exit itsoME with option RTQM=GWQM/, the following flow will occur (these numbers correspond to the numbers in Figure A-2):

1. APP1 sends a message intended for BPQM1 and sets ReplyToQ=REPLY on QM1.
2. itsoME changes ReplyToQmgr to GWMQ, and changes the ReplyToQ=QM1.REPLY.
3. The message is sent to BPQM1 and processed. The reply comes back to GWMQ on QM1.REPLY.
4. GWQM uses its QREMOTE(QM1.REPLY) definition that points back to QM1.

A.3.1 Configuring the queues

Example A-4 shows the commands to define the queue definitions for queue managers QM1, QM2, and QM3.

Example A-4 Definitions in QM1, QM2, and QM3

```
DEFINE QLOCAL(REPLY)

DEFINE QLOCAL(GWQM) USAGE(XMITQ)....

DEFINE QREMOTE(SERVICE) RNAME(GWMQ) XMITQ(GWMQ)...
```

Example A-5 shows the commands to define the queue definitions for queue manager GWQM.

Example A-5 Definitions in GWQM

```
DEFINE QREMOTE(QM1.REPLY) RNAME(REPLY) RQMNAME(QM1) XMITQ(QM1)...

DEFINE QREMOTE(QM2.REPLY) RNAME(REPLY) RQMNAME(QM2) XMITQ(QM2)...

DEFINE QREMOTE(QM3.REPLY) RNAME(REPLY) RQMNAME(QM3) XMITQ(QM2)...

DEFINE QREMOTE(SERVICE) RNAME(SERVICE) RQMNAME(BPQM1) XMITQ(BPQM1)...

ALTER CHANNEL(GWQM.BPQM1) CHLTYPE(SDR) +
      MSGEXIT('itsoME(MsgExit)') +
      MSGDATA('RTQM=GWQM/')

DEFINE QLOCAL(QM1) USAGE(XMITQ)...

DEFINE QLOCAL(QM2) USAGE(XMITQ)...

DEFINE QLOCAL(QM3) USAGE(XMITQ)...

DEFINE QLOCAL(BPQM1) USAGE(XMITQ)...
```

RTMQ=: Message exit itsoME option RTMQ= will not change the ReplyToQ if the ReplyToQmgr matches the specified queue manager.

A.4 Configuring the exit

As you can see in Example A-2 on page 266 and Example A-5, the exit is defined on the channel definition:

```
DEFINE CHANNEL(GWQM.BPQM1) MSGEXIT('itsoME(MsgExit)') MSGDATA('RTQM=GWQM/')
```

The syntax for MSGDATA is:

```
MSGDATA('Option/[Option/][Option/]')
```

All options are followed by a forward slash '/' to separate the commands.

The following options are supported in itsoME.

Table A-1 Options supported by message exit itsoME

Option	Channel type	Description
ALL	RCVR/RQSTR	Removes COA/EXP/DISC options
COA	RCVR/RQSTR	Removes the COA and COD report options
DISC	RCVR/RQSTR	Removes discard options to limit resource enumeration attacks. Mis-addressed messages are delivered to the dead-letter queue (DLQ)
EXP	RCVR/RQSTR	Removes Exception and Expiry report options
MCA	RCVR/RQSTR	Sets the received UserIdentifier in Message Descriptor to the value of the MCAUSER field from the channel definition
NONE	RCVR/RQSTR	Removes COA/EXP/DISC options
RTQM=	RCVR/RQSTR SDR/SVR	Prefix ReplyToQueue with ReplyToQmgr'.'

A.5 Compiling the exit

The itsoME message exit that was developed for the scenarios in this book is provided as source to use as a starting point for your own exit. Before using the exit, you must compile the source.

A.5.1 z/OS

A sample job to compile the source on z/OS is shown in Example A-6.

Example A-6 JCL used to compile message exit ITSOME

```
//ITSOMEME JOB
//*******************************************************************
//*                                                                 *
//*  BUILD ITSOME FOR MQ VERSION 7.1 ON Z/OS 1.12                   *
//*                                                                 *
//*                                                                 *
//*******************************************************************
//*                                                                 *
//* LICENSED MATERIALS - PROPERTY OF IBM                            *
//*                                                                 *
//* 5645-001                                                        *
//* (C) COPYRIGHT IBM CORP. 1988, 1997 ALL RIGHTS RESERVED          *
//*                                                                 *
//* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                    *
//* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                 *
//* SCHEDULE CONTRACT WITH IBM CORP                                 *
//*                                                                 *
//*******************************************************************
//*                                                                 *
//*    Replace   ++YOURSRC++                                        *
//*                        with the name of your exit source data set. *
//*                                                                 *
```

```
//*    Replace   ++THLQUAL++                                        *
//*                      with the high level qualifier of the       *
//*                      WebSphere MQ target library data sets.     *
//*                                                                 *
//*    Replace   ++EXITLIB++                                        *
//*                      with the name of your user exit data set.  *
//*                                                                 *
//*    Replace   ++LEQUAL++                                         *
//*                      with the high level qualifier of the       *
//*                      LE Library target library data sets.       *
//*                                                                 *
//*******************************************************************
//*
//EDCC  PROC  INFILE=,                  < INPUT ... REQUIRED
//  CREGSIZ='0M',                       < COMPILER REGION SIZE
//  CRUN=,                              < COMPILER RUNTIME OPTIONS
//  CPARM=,                             < COMPILER OPTIONS
//  CPARM2=,                            < COMPILER OPTIONS
//  CPARM3=,                            < COMPILER OPTIONS
//  SYSLBLK='3200',                     < BLOCKSIZE FOR &&LOADSET
//  LIBPRFX='CEE',                     < PREFIX FOR LIBRARY DSN
//  LNGPRFX='CCPLUS',              < PREFIX FOR LANGUAGE DSN
//  CLANG='EDCMSGE', < NOT USED IN THIS RELEASE. KEPT FOR COMPATIBILIT
//  DCB80='(RECFM=FB,LRECL=80,BLKSIZE=3200)',      <DCB FOR LRECL 80
//  DCB3200='(RECFM=FB,LRECL=3200,BLKSIZE=12800)', <DCB FOR LRECL 3200
//  OUTFILE='&&LOADSET,DISP=(MOD,PASS),UNIT=VIO,SPACE=(TRK,(7,7))'
//*
//*-----------------------------------------------------------------
//*  COMPILE STEP:
//*-----------------------------------------------------------------
//COMPILE EXEC PGM=CCNDRVR,REGION=&CREGSIZ,
//    PARM=('&CRUN/&CPARM &CPARM2 &CPARM3')
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//         DD  DSNAME=&LIBPRFX..SCEERUN2,DISP=SHR
//SYSMSGS  DD  DUMMY,DSN=&LNGPRFX..SCBC3MSG(&CLANG),DISP=SHR
//SYSIN    DD  DSNAME=&INFILE,DISP=SHR
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEEH.H,DISP=SHR
//         DD  DSNAME=&LIBPRFX..SCEEH.SYS.H,DISP=SHR
//         DD  DSNAME=&LIBPRFX..SCEEH.NETINET.H,DISP=SHR
//SYSLIN   DD  DSNAME=&OUTFILE,DCB=(RECFM=FB,LRECL=80,BLKSIZE=&SYSLBLK)
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSCPRT  DD  SYSOUT=*
//SYSUT1   DD  UNIT=VIO,SPACE=(32000,(30,30)),DCB=&DCB80
//SYSUT4   DD  UNIT=VIO,SPACE=(32000,(30,30)),DCB=&DCB80
//SYSUT5   DD  UNIT=VIO,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT6   DD  UNIT=VIO,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT7   DD  UNIT=VIO,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT8   DD  UNIT=VIO,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT9   DD  UNIT=VIO,SPACE=(32000,(30,30)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10  DD  SYSOUT=*
//SYSUT14  DD  UNIT=VIO,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//         PEND
```

```
//*
//*******************************************************************
//*                                                                 *
//* LICENSED MATERIALS - PROPERTY OF IBM                            *
//*                                                                 *
//* 5688-188 (C) COPYRIGHT IBM CORP. 1993                           *
//* ALL RIGHTS RESERVED                                             *
//*                                                                 *
//* US GOVERNMENT USERS RESTRICTED RIGHTS - USE,                    *
//* DUPLICATION OR DISCLOSURE RESTRICTED BY GSA ADP                 *
//* SCHEDULE CONTRACT WITH IBM CORP                                 *
//*                                                                 *
//* SEE COPYRIGHT INSTRUCTIONS                                      *
//*                                                                 *
//*******************************************************************
//*                                                                 *
//* IBM C/370 VERSION 2 RELEASE 2                                   *
//*                                                                 *
//* EDCPL   --- THIS CATALOGUED PROCEDURE PRE-LINKS                 *
//*  (REQUIRED FOR REENTRANCY), AND LINK-EDITS A C PROGRAM          *
//*                                                                 *
//*                                                                 *
//*  PARAMETER  DEFAULT VALUE    USAGE                              *
//*   INFILE    NONE             INPUT DATA SET                     *
//*   OUTFILE   &&GSET(GO)       OUTPUT LINK EDIT DATA SET          *
//*   SYSLBLK   3200             BLOCKSIZE FOR &&PLKSET             *
//*   PREGSIZ   2048K            PRE-LINKER REGION SIZE             *
//*   PPARM     NONE             PRE-LINKER OPTIONS                 *
//*   PLIB      DUMMY            PRELINK AUTOCALL LIBRARY           *
//*   LREGSIZ   1024K            LINK EDIT REGION SIZE              *
//*   LPARM     AMODE=31,MAP     LINK EDIT OPTIONS                  *
//*   LIBPRFX   EDC.V2R2M0       PREFIX FOR LIBRARY DATA SET NAMES  *
//*   COMPRFX   PLI.V2R3M0       PREFIX FOR COMMON LIBRARY          *
//*                                                                 *
//*******************************************************************
//*
//EDCPL  PROC INFILE=,
// OUTFILE='&&GSET(GO),DISP=(MOD,PASS),UNIT=VIO,SPACE=(512,(50,20,1))',
//   SYSLBLK='3200',
//   PREGSIZ='2048K',
//   PPARM=,
//   PLIB='DUMMY',
//   LREGSIZ='1024K',
//   LPARM='AMODE=31,MAP',
//   LIBPRFX='CEE'
//*
//*-------------------------------------------------------------
//* PRE-LINKEDIT STEP:
//*-------------------------------------------------------------
//PLKED   EXEC PGM=EDCPRLK,PARM='&PPARM',
//    REGION=&PREGSIZ
//SYSMSGS  DD  DSNAME=&LIBPRFX..SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB   DD  &PLIB
//SYSIN    DD  DSN=&INFILE,DISP=SHR
//SYSMOD   DD  DSNAME=&&PLKSET,UNIT=VIO,DISP=(MOD,PASS),
```

```
//             SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=80,BLKSIZE=&SYSLBLK)
//SYSOUT   DD  SYSOUT=*
//SYSPRINT DD  SYSOUT=*
//*
//*-------------------------------------------------------------------
//* LINKEDIT STEP:
//*-------------------------------------------------------------------
//LKED   EXEC PGM=IEWBLINK,COND=(4,LT,PLKED),
//    REGION=&LREGSIZ,PARM='&LPARM'
//SYSLIB   DD  DSNAME=&LIBPRFX..SEDCBASE,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSLIN   DD  DSNAME=*.PLKED.SYSMOD,DISP=SHR
//         DD  DDNAME=SYSIN
//SYSLMOD  DD  DSNAME=&OUTFILE
//SYSUT1   DD  UNIT=VIO,SPACE=(32000,(30,30))
//         PEND
//*
//*
//CCOMPR   EXEC PROC=EDCC,       /* C/MVS   */
//         CPARM='SOURCE,LONGNAME,SS,SHOWINC,NOSEQUENCE,NOMARGINS,RENT'
//*
//COMPILE.SYSLIB  DD
//                DD
//                DD
//                DD DSN=++THLQUAL++.SCSQC370,DISP=SHR
//                DD DSN=++LEQUAL++.SCEESAMP,DISP=SHR
//*
//COMPILE.SYSIN   DD DSN=++YOURSRC++(ITSOME),DISP=SHR
//*
//CLINKR   EXEC PROC=EDCPL,COND=(4,LT),
//         PLIB='DSN=++LEQUAL++.SCEECPP,DISP=SHR',
//         LPARM='AMODE=31,MAP,REUS(RENT)',
//         INFILE=&&LOADSET,
//         OUTFILE='++EXITLIB++,DISP=SHR'
//LKED.SYSLIB   DD DSN=++LEQUAL++.SCEESPC,DISP=SHR
//              DD DSN=++LEQUAL++.SCEERUN,DISP=SHR
//              DD DSN=++LEQUAL++.SCEELKEX,DISP=SHR
//              DD DSN=++LEQUAL++.SCEELKED,DISP=SHR
//              DD DSN=++THLQUAL++.SCSQLOAD,DISP=SHR
//              DD DSN=SYS1.CSSLIB,DISP=SHR
//LKED.SYSIN    DD *
    ENTRY ITSOME
    NAME ITSOME(R)
//*
```

A.5.2 Linux for System z

Example A-7 shows how to compile the source on Linux for IBM System z®.

Example A-7 Command to compile message exit itsoME on Linux for System z

```
gcc -m64 -shared -fPIC -o itsoME itsoME.c -I/opt/mqm/inc -Wl
```

A.5.3 Linux 64

Example A-8 shows how to compile the source on Linux 64-bit systems.

Example A-8 Command to compile message exit itsoME on Linux 64 bit

```
gcc  -m64 -o itsoME itsoME.c -fPIC -ansi -shared -I/opt/mqm/inc
```

A.5.4 Linux 32

Example A-9 shows how to compile the source on Linux 32-bit systems.

Example A-9 Command to compile message exit itsoME on Linux 32 bit

```
gcc  -o itsoME itsoME.c -fPIC -ansi -shared -I/opt/mqm/inc -Wall
```

A.5.5 AIX

Example A-10 shows how to compile the source on IBM AIX® systems.

Example A-10 Command to compile message exit itsoME on AIX

```
xlc_r -q64 -e MQStart -bE:itsoME.exp -o itsoME itsoME.c -I/usr/mqm/inc
```

Example A-11 shows the content of itsoME.exp.

Example A-11 `itsoME.exp content`

```
MsgExit
MQStart
```

A.5.6 Solaris SPARC

Example A-12 shows how to compile the source on SPARC systems.

Example A-12 Command to compile message exit itsoME on Solaris SPARC

```
gcc -I/opt/mqm/inc -m64 -o itsoME itsoME.c -G -I/opt/mqm/inc -lc
```

A.5.7 Microsoft Windows

Example A-13 shows how to compile the source on Windows systems.

Example A-13 Command to compile message exit itsoME on Windows

```
cl /LD  itsoME.c -D_CRT_SECURE_NO_DEPRECATE
```

A.6 Installing the exit

This section describes the configuration that is required to activate the exit.

A.6.1 Installation on z/OS

On z/OS, the message exit must be placed in a load library that is allocated to the CSQXLIB DD statement in your <*qmgr*>CHIN task as described in the section “Writing channel exit programs on z/OS” in the WebSphere MQ Information Center:

http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp?topic=%2Fcom.ibm.mq.doc%2Fjm34740_.htm

Then, you can add the exit to your channel with the WebSphere MQ **ALTER CHANNEL** command.

Example A-14 Activation of ITSOME message exit on z/OS

```
ALTER CHANNEL(CSQ1.TO.CSQ2) CHLTYPE(RCVR) +
   MSGEXIT('ITSOME') +
   MSGDATA('MCA/NONE/')
```

A.6.2 Installation on distributed platforms

This installation assumes that the compiled version of itsoME is in the default exit folder for WebSphere MQ.

Then, you can add the exit to your channel with the WebSphere MQ **ALTER CHANNEL** command:

Example A-15 Activation of itsoME message exit on a distributed platform

```
ALTER CHANNEL(CSQ1.TO.CSQ2) CHLTYPE(RCVR) +
   MSGEXIT('itsoME(MsgExit)') +
   MSGDATA('MCA/NONE/')
```

A.7 Design considerations

Message exit itsoME has deliberately been kept simple. While more could have been done to make it more flexible or perform better, the goal is to keep it as simple as possible and still fulfill the need. Therefore, the pExitBufferAddr pointer that is provided by WebSphere MQ is used to hold various flags instead of obtaining memory that would live after the environment is terminated on z/OS and IBM i.

This implementation will write one line in a log when the channel is started and one line when the channel ends. In the case of a syntax or parameter error in specifying the MSGDATA option, it will write an additional line.

This log is written to these locations:

- On AIX, Linux, and Solaris: `/var/mqm/exits/ITSOPWE.log`
- On z/OS: DDNAME SYSPRINT
- On Windows: program event log

Example A-16 on page 275 shows the log entries that are created when the `MSGDATA` parameter has been specified correctly.

Example A-16 Normal example where the MSGDATA parameter is correct

```
itsoME: ver=1.00 Channel="CSQ1.QM2" started with options="RTQM=PETER/"

itsoME: ver=1.00 ChannelName="CSQ1.QM2" ended.
```

Example A-17 shows the log entries that are created when the `MSGDATA` parameter has not been specified correctly (the trailing / is missing).

Example A-17 Trailing "/" is missing from MSGDATA

```
itsoME: ver=1.00 Channel="CSQ1.QM2" started with options="RTQM=PETER"
itsoME: Trailing / is missing in MSGDATA. Channel is stopped.
itsoME: ver=1.00 ChannelName="CSQ1.QM2" ended.
```

Example A-18 shows another example of the log entries when the `MSGDATA` parameter is specified incorrectly.

Example A-18 An additional option "s" was specified

```
itsoME: ver=1.00 Channel="CSQ1.QM2" started with options="MCA/COA/RTQM=OLE/s/"
itsoME: Invalid option was specified="s" Channel is stopped.
itsoME: ver=1.00 ChannelName="CSQ1.QM2" ended.
```

A.8 Debugging

There is some built-in debugging help in the exit. To activate the debugging option during compile time, define the symbol DEBUGMSG. Set DEBUGMSG=1 to send the debug messages to STDOUT. Set DEBUGMSG=2 to send the output to a file. The name of the file is defined in the source code.

Example A-19 shows how to set the debug option during compilation on a Windows system.

Example A-19 Command to compile message exit itsoME on Windows with debug enabled

```
cl /LD  itsoME.c -D_CRT_SECURE_NO_DEPRECATE -DDEBUGMSG=2
```

If you would like to debug on z/OS, the easiest way is to change the `#define DEBUGMSG 0` line in the source to `#define DEBUGMSG 2`.

Limitations

Message exit itsoME has a limit on the length of the supported reply to queue names. The maximum queue name is 48 characters, which means that specifying a remote queue manager name of RTQM=GWQM1 reduces the maximum length of the reply to queue name by six characters (48 - (5 + 1)). itsoME will truncate the resulting Reply To queue name to 48 characters.

A.9 Message exit source

Example A-20 on page 276 contains the source for the sample exit.

Example A-20 Source code for message exit itsoME

```
/*****************************************************************************
 * itsoME.c -  message exit to modify received messages to prevent security
 *             problems when using COA, COD, Expiry messages by modifying the
 *             received userid.
 *             The exit can used to change the reply to queue manager in sent or
 *             received messages and set it to a value that suits your
 *             installation to increase Security.
 *             For usage see IBM Redbooks publication SG-xxxx-xx available from:
 *             http://www.redbooks.ibm.com/
 *
 * ---------------------------------------------------------------------------
 * Description
 *
 * This message exit is designed to remove replace the received userid with the
 * MCAUSER defined on the channel definition to prevent security violations when
 * forwarding message thru an immediate queue manager.
 *
 * The exit needs to be added to the channel definitions on the RCVR/RQSTR
 * channel.
 *
 * Compilation:
 * Windows:
 *   cl /LD  itsoME.c -D_CRT_SECURE_NO_DEPRECATE Advapi32.lib
 *
 * Linux (32):
 *   gcc  -o itsoME itsoME.c -fPIC -ansi -shared -I/opt/mqm/inc -Wall
 *
 * Linux (64):
 *   gcc  -m64 -o itsoME itsoME.c -fPIC -ansi -I/opt/mqm/inc -shared
 *
 * Linux (z):
 *   gcc -m64 -shared -fPIC -o itsoME itsoME.c -I/opt/mqm/inc -Wl
 *
 * Solaris SPARC:
 *   gcc -I/opt/mqm/inc -m64 -o itsoME itsoME.c -G -lc
 *
 * AIX (64bit):
 * xlc_r -q64 -e MQStart -bE:itsoME.exp -o itsoME itsoME.c -I/usr/mqm/inc
 * itsoME.exp contains:
MsgExit
MQStart
 *
 * ---------------------------------------------------------------------------
 * History
 *
 * Written by Jørgen Pedersen                            - 01-Jul12 DK  v1.00
 * email: jopde@dk.ibm.com
 *
 * ---------------------------------------------------------------------------
 * Documentation
 *
 *
 * ---------------------------------------------------------------------------
 * Channel definitions
```

```
 *
 * To add msgexit to channel definitions distributed queue managers:
 *   alt chl(MQT2.TCP.MQT1) chltype(RCVR) +      * OR: RQSTR/SDR/SVR
 *   msgexit('itsoME(MsgExit)') +                *
 *   msgdata('MCA/NONE/')                        *
 *
 * z/os:
 *   alt chl(MQT2.TCP.MQT1) chltype(RCVR) +      * OR: RQSTR/SDR/SVR
 *   msgexit('ITSOME') +                         *
 *   msgdata('MCA/NONE/')                        *
 *
 * Usage:
 * syntax: MSGDATA('Option/[Option/][Option/]')
 * options:
 * MCA   - Setting received UserIdentifier in Message Descriptor to the
 *         value of MCAUser field from the channel
 * COA   - removes the COA and COD report options
 * EXP   - removes Exception and Expiry report options
 * DISC  - removes discard options preventing snooping. Msgs will go into DLQ
 * ALL   - clears COA/EXP/DISC options off
 * NONE  - clears COA/EXP/DISC options off
 * RTQM= - Prefix ReplyToQueue with ReplyToQmgr'.'
 *
 * Example to remove COA and set UserIdentifer to MCA value:
 * msgdata('COA/MCA/')
 * Remember the trailing slash '/' otherwise the option will be ignored.
 */

/*****************************************************************************
 * -- environment --
 *****************************************************************************/

/* define to get debug message output, either below or in compile cmd line      */
#if !defined(DEBUGMSG)
#define DEBUGMSG   0                /* 0 - no output, 1 - stdout, 2 - file out */
#endif

#define SRCID "@(#)itsoME.c 1.01 DATE4-Jul-2012"
#define CPRID "@(#)Copyright 2012 IBM"
#define OWNID "@(#)Licensed Materials - Property of IBM"

/*****************************************************************************
 * -- Includes --
 *****************************************************************************/

/* standard headers                                                          */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

/* MQ headers                                                                */
#include <cmqc.h>
#include <cmqxc.h>
```

```
#if (MQAT_DEFAULT==MQAT_WINDOWS_NT)
#include <Windows.h>
#endif
/******************************************************************************
 * -- defines & macros --
 ******************************************************************************/

#if (MQAT_DEFAULT==MQAT_WINDOWS_NT)
#define EXPORT __declspec(dllexport)    /* exported from DLL */
#else
#define EXPORT MQENTRY
#endif  /* MQAT_WINDOWS_NT */

#if (MQAT_DEFAULT==MQAT_ZOS)
/* MVS specific code */
  #pragma runopts(nospie,stae)
  #pragma environment(ITSOME)
  #include <spc.h>
#endif /* MQAT_ZOS */

/* Options for controlling operation                                        */
#define MQME_NONE  0x00000000
#define MQME_ALL   0x00000007
#define MQME_COA   0x00000001
#define MQME_EXP   0x00000002
#define MQME_DISC  0x00000004
#define MQME_MCA   0x00001000
#define MQME_RTQM  0x00002000

#define VERSION      "1.00"                        /* Version no             */

#if MQAT_DEFAULT==MQAT_WINDOWS_NT
  #define MSGFILE   "c:\\ITSOPWE.log" /* <-- tailor for environment */
#endif
#if !defined MSGFILE
  #define MSGFILE   "/var/mqm/exits/ITSOPWE.log"   /* <-- tailor for environment
*/
#endif

/******************************************************************************
 *
 * debug message generation
 *
 ******************************************************************************/

#if DEBUGMSG == 1
/* debug messages to stdout */
  #define OST stdout
  #define DBGPRINTF(x) fprintf x
#elif DEBUGMSG == 2

#if MQAT_DEFAULT==MQAT_ZOS
  #define DBGPRINTF(x) { \
                       OST = trcbuf ; \
                       sprintf x ; \
```

```
                        i=strlen(OST) ; \
                        printf(OST); \
                        }
#else
FILE * OST = 0;
  #define DBGPRINTF(x) { \
                        OST=fopen(MSGFILE,"a+"); \
                        if(OST) fprintf x; \
                        fclose(OST); \
                        }
#endif

#else
/* no debug messages */
  #define DBGPRINTF(x)
#endif /* DEBUGMSG */

/* Macro used for logging hello, bye and errors                                  */
#if (MQAT_DEFAULT==MQAT_ZOS)
  #define DBGLOG(x) { \
                        OSL = trcbuf ; \
                        sprintf x ; \
                        i=strlen(OSL) ; \
                        printf(OSL); \
                        }
#else
#if (MQAT_DEFAULT==MQAT_WINDOWS_NT)
  #define DBGLOG(x) { \
                        OSL = trcbuf ; \
                        sprintf x ; \
                        SendLogMessage(OSL); \
                        }
#else
FILE * OSL = 0;
  #define DBGLOG(x) { \
                        OSL=fopen(MSGFILE,"a+"); \
                        if(OSL) fprintf x; \
                        fclose(OSL); }
#endif
#endif

#if MQAT_DEFAULT==MQAT_WINDOWS_NT
void SendLogMessage(char *szData);
#endif

#if !defined (_REENTRANT)
char *strtok_r(char * s, const char * delim , char ** ptrptr );
#endif

#if !defined (_REENTRANT)
char *strtok_r(char * s, const char * delim , char ** ptrptr )
{
    return(strtok(s, delim));
}
#endif
```

```
/*****************************************************************************
 * rlnts()
 *
 * Remove trailing spaces
 *
 *
 * @return length, string has no trailing spaces
 *****************************************************************************/
static int rlnts(int l, char * s)
{
    while( l && s[l-1] == ' ') --l;
    return l;
}


static char SrcIdent[] = SRCID, CprIdent[] = CPRID, OwnIdent[] = OWNID;

void strlower(char* string, int string_length)
{
    int length = string_length < 0 ? strlen(string) : string_length;
    while(length--)
    {
        *string = tolower(*string);
        string++;
    }
}

void strupper(char* string, int string_length)
{
    int length = string_length < 0 ? strlen(string) : string_length;
    while(length--)
    {
        *string = toupper(*string);
        string++;
    }
}

/*****************************************************************************
 * MQExitReason()
 *
 * Return ExitReasonText string
 *
 *
 * @return exit Reason text.
 *****************************************************************************/
char * MQExitReason(PMQCXP pChannelExitParams)
{
    switch(pChannelExitParams->ExitReason)
    {
    case(MQXR_INIT):            return("MQXR_INIT");
    case(MQXR_TERM):            return("MQXR_TERM");
    case(MQXR_MSG):             return("MQRX_MSG");
    case(MQXR_XMIT):            return("MQXR_XMIT");
    case(MQXR_SEC_MSG):         return("MQXR_SEC_MSG");
```

```
    case(MQXR_INIT_SEC):        return("MQXR_INIT_SEC");
    case(MQXR_RETRY):           return("MQXR_RETRY");
    case(MQXR_AUTO_CLUSSDR):    return("MQXR_AUTO_CLUSSDR");
    case(MQXR_AUTO_RECEIVER):   return("MQXR_AUTO_RECEIVER");
    case(MQXR_AUTO_SVRCONN):    return("MQXR_AUTO_SVRCONN");
    case(MQXR_AUTO_CLUSRCVR):   return("MQXR_AUTO_CLUSRCVR");
#ifdef MQXR_SEC_PARMS
    case(MQXR_SEC_PARMS):         return("MQXR_SEC_PARMS");
#endif
    default:                    return("Unknown ExitReason");
    }                                           /* end switch              */
}

/******************************************************************************
 * MQExitId()
 *
 * Return ExitIdText string
 *
 *
 * @return exit Id text.
 ******************************************************************************/
char * MQExitId(PMQCXP pChannelExitParams)
{
    switch(pChannelExitParams->ExitId)
    {
    case(MQXT_CHANNEL_SEC_EXIT):        return("MQXT_CHANNEL_SEC_EXIT");
    case(MQXT_CHANNEL_MSG_EXIT):        return("MQXT_CHANNEL_MSG_EXIT");
    case(MQXT_CHANNEL_SEND_EXIT):       return("MQXT_CHANNEL_SEND_EXIT");
    case(MQXT_CHANNEL_RCV_EXIT):        return("MQXT_CHANNEL_RCV_EXIT");
    case(MQXT_CHANNEL_MSG_RETRY_EXIT):  return("MQXT_CHANNEL_MSG_RETRY_EXIT");
    case(MQXT_CHANNEL_AUTO_DEF_EXIT):   return("MQXT_CHANNEL_AUTO_DEF_EXIT");
    default:                            return("Unknown ExitID");
    }                                           /* end switch              */
}

/******************************************************************************
 * MQChannelType()
 *
 * Return ChannelTypeText string
 *
 *
 * @return exit ChannelType text.
 ******************************************************************************/
char * MQChannelType(PMQCD pChannelDefinition)
{
    switch(pChannelDefinition->ChannelType)
    {
    case(MQCHT_SENDER):         return("MQCHT_SENDER");
    case(MQCHT_SERVER):         return("MQCHT_SERVER");
    case(MQCHT_RECEIVER):       return("MQCHT_RECEIVER");
    case(MQCHT_REQUESTER):      return("MQCHT_REQUESTER");
    case(MQCHT_CLNTCONN):       return("MQCHT_CLNTCONN");
    case(MQCHT_SVRCONN):        return("MQCHT_SVRCONN");
    case(MQCHT_CLUSSDR):        return("MQCHT_CLUSSDR");
    case(MQCHT_CLUSRCVR):       return("MQCHT_CLUSRCVR");
```

```
    default:                    return("Unknown ChannelType");
    }                                           /* end switch              */
}

/*****************************************************************************
 * MQExitResponseType()
 *
 * Return ExitResponseText string
 *
 *
 * @return exit Response text.
 *****************************************************************************/
char * MQExitResponseType(PMQCXP pChannelExitParams)
{
    switch(pChannelExitParams->ExitResponse)
    {
    case(MQXCC_OK):                     return("MQXCC_OK");
    case(MQXCC_SUPPRESS_FUNCTION):      return("MQXCC_SUPPRESS_FUNCTION");
    case(MQXCC_SKIP_FUNCTION):          return("MQXCC_SKIP_FUNCTION");
    case(MQXCC_SEND_AND_REQUEST_SEC_MSG):
return("MQXCC_SEND_AND_REQUEST_SEC_MSG");
    case(MQXCC_SEND_SEC_MSG):           return("MQXCC_SEND_SEC_MSG");
    case(MQXCC_SUPPRESS_EXIT):          return("MQXCC_SUPPRESS_EXIT");
    case(MQXCC_CLOSE_CHANNEL):          return("MQXCC_CLOSE_CHANNEL");
    case(MQXCC_FAILED):                 return("MQXCC_FAILED");
    default:                            return("Unknown ExitResponse");
    }                                           /* end switch              */
}

/*****************************************************************************
 * SendLogMessage()
 *
 *
 * Method to log data to EventList on windows
 *
 *  szData ASCII data to report
 *
 * @return void
 *
 *****************************************************************************/
#if MQAT_DEFAULT==MQAT_WINDOWS_NT
void SendLogMessage(char *szData)
{
    HANDLE  hSource;
    char    *szList[1];

    szList[0]=szData;
    hSource = RegisterEventSource(NULL, "IBM itsoME");
    if (hSource != NULL)
    {
        ReportEvent(hSource,EVENTLOG_INFORMATION_TYPE,
                    0,1,NULL,1,0,(const unsigned char**)szList,NULL);
        DeregisterEventSource(hSource);
    }
}
```

```
#endif

/******************************************************************************/

extern void MQENTRY MQStart(void) {;}

/******************************************************************************
 * MsgExit - Channel message exit, entry point
 *
 * PARAMETERS:
 *    pChannelExitParams (PMQCXP)  - input/output
 *       Channel exit parameter block
 *
 *    pChannelDefinition (PMQCD)   - input/output
 *       Channel definition
 *
 *    pDataLength        (PMQLONG) - input/output
 *       On input:  length of AgentBuffer
 *       On output: length of returned data (AgentBuff/ErrorBuff)
 *
 *    pAgentBufferLength (PMQLONG) - input
 *       Agent Buffer length
 *
 *    AgentBuffer        (PMQBYTE) - input/output
 *       Agent buffer
 *
 *    pExitBufferLength  (PMQLONG) - input/output
 *       Exit buffer length
 *
 *    pExitBufferAddr    (PMQPTR)  - input/output
 *       Exit buffer
 *
 * RETURN:
 *    void
 *
 * NOTES:
 *    pExitBufferAddr
 *       This pointer is used to hold switches for communication between various
 *       exit points. This was done for simplicity of this exit to provide a
 *       broader solution that will work on most platforms.
 *
 ******************************************************************************/
#if (MQAT_DEFAULT==MQAT_ZOS)
void EXPORT ITSOME(  PMQCXP     pChannelExitParams,
                     PMQCD      pChannelDefinition,
                     PMQLONG    pDataLength,
                     PMQLONG    pAgentBufferLength,
                     PMQBYTE    AgentBuffer,
                     PMQLONG    pExitBufferLength,
                     PMQPTR     pExitBufferAddr)
#else
void EXPORT MsgExit( PMQCXP     pChannelExitParams,
                     PMQCD      pChannelDefinition,
                     PMQLONG    pDataLength,
                     PMQLONG    pAgentBufferLength,
```

```
                    PMQBYTE    AgentBuffer,
                    PMQLONG    pExitBufferLength,
                    PMQPTR     pExitBufferAddr)
#endif
{
    MQXQH * pXqh = (MQXQH *)AgentBuffer;

    MQCHAR  cExitDataWork[MQ_EXIT_DATA_LENGTH+1+5];
    MQCHAR  cWorkRQmgr[MQ_Q_MGR_NAME_LENGTH+1+5];
    MQCHAR  cWorkRQmgr1[MQ_Q_MGR_NAME_LENGTH+1+5];
    MQCHAR  cWorkReplyQueue[MQ_Q_MGR_NAME_LENGTH+MQ_Q_NAME_LENGTH+2];
    PMQCHAR p,u;
    MQLONG  lOptions = (MQLONG)MQME_NONE;
    char   *strtok_cb;

#if (MQAT_DEFAULT==MQAT_ZOS) || (MQAT_DEFAULT==MQAT_WINDOWS_NT)
    unsigned char trcbuf[255] ;
    unsigned char * OSL ;
#endif
#if (MQAT_DEFAULT==MQAT_ZOS)
    long     i;
    unsigned char * OST ;
#endif

    /* keep compiler quiet                                                  */
    pExitBufferAddr   = pExitBufferAddr;
    pExitBufferLength = pExitBufferLength;

    /* debug info                                                           */
    DBGPRINTF((OST,"itsoME: ver=%s ExitId=%s (%ld) ExitReason=%s (%ld) "
               "ChannelType=%s (%d)\n",
               VERSION,
               MQExitId(pChannelExitParams),
               pChannelExitParams->ExitId,
               MQExitReason(pChannelExitParams),
               pChannelExitParams->ExitReason,
               MQChannelType(pChannelDefinition),
               pChannelDefinition->ChannelType));

    DBGPRINTF((OST,"itsoME: QMgrName=\"%.*s\"\n",
               rlnts(MQ_Q_MGR_NAME_LENGTH, pChannelDefinition->QMgrName),
               pChannelDefinition->QMgrName));

    DBGPRINTF((OST,"itsoME: ChannelName=\"%.*s\"\n",
               rlnts(MQ_CHANNEL_NAME_LENGTH, pChannelDefinition->ChannelName),
               pChannelDefinition->ChannelName));

    /* set default exit responses                                           */
    pChannelExitParams->ExitResponse  = MQXCC_OK;
    pChannelExitParams->ExitResponse2 = MQXR2_USE_AGENT_BUFFER;

    /* ensure we are only being called as a channel msgexit                 */
    if ( pChannelExitParams->ExitId != MQXT_CHANNEL_MSG_EXIT )
    { /* error - not being invoked from msgexit */
        DBGPRINTF((OST,"itsoME: error - exit called in wrong context\n\n"));
```

```
        /* prevent exit being called again                                  */
        pChannelExitParams->ExitResponse = MQXCC_SUPPRESS_EXIT;
        return;
    }

    /* switch on ExitReason to see why exit was invoked                     */
    switch(pChannelExitParams->ExitReason)
    {

    case MQXR_INIT: /* Initialization */
        DBGLOG((OSL,
                "itsoME: ver=%s Channel=\"%.*s\" started with options=\"%.*s\"\n",
                VERSION,
                rlnts(MQ_CHANNEL_NAME_LENGTH, pChannelDefinition->ChannelName),
                pChannelDefinition->ChannelName,
                rlnts(MQ_EXIT_DATA_LENGTH, pChannelDefinition->MsgUserData),
                pChannelDefinition->MsgUserData));

        DBGPRINTF((OST,"itsoME: init\n"));

        /* No options selected yet. Do nothing is default                   */
        /* NOTE: pExitBufferAddr is used to pass flags between exit points  */
        *pExitBufferAddr = (PMQPTR)MQME_NONE;

        /* prepare for parsing MsgUserData */
        memset(cExitDataWork,'\00',sizeof(cExitDataWork));
        memcpy(cExitDataWork, pChannelDefinition->MsgUserData,
               rlnts(MQ_EXIT_DATA_LENGTH, pChannelDefinition->MsgUserData));

        /* Check for trailing "/"                                           */
            if (cExitDataWork[strlen(cExitDataWork)-1] != '/')
            {                                       /* option, full-stop    */
                DBGLOG((OSL,
                "itsoME: Trailing / is missing in MSGDATA. Channel is
stopped.\n"));
                pChannelExitParams->ExitResponse  = MQXCC_CLOSE_CHANNEL;
                pChannelExitParams->ExitResponse2 = MQXR2_USE_AGENT_BUFFER;
                break;
            }

        /* processing the MsgUserData to find wanted options                */
        p = (char*)strtok_r(cExitDataWork, "/", &strtok_cb);
        while (p!=NULL)
        {
            if (!strncmp(p,"ALL",3))
                lOptions = lOptions | MQME_ALL;
            else if (!strncmp(p,"COA",3))
                lOptions = lOptions | MQME_COA;
            else if (!strncmp(p,"DLQ",3))
                lOptions = lOptions | MQME_MCA;
            else if (!strncmp(p,"EXP",3))
                lOptions = lOptions | MQME_EXP;
            else if (!strncmp(p,"MCA",3))
                lOptions = lOptions | MQME_MCA;
```

```
            else if (!strncmp(p,"NONE",4))
                lOptions = lOptions | MQME_ALL;
            else if (!memcmp(p,"RTQM=",5))          /* we need to modify       */
                lOptions = lOptions | MQME_RTQM;     /*  reply to queue         */
            else                                     /* we have a non supported */
            {                                        /* option, full-stop       */
                DBGLOG((OSL,
                "itsoME: Invalid option was specified=\"%s\" Channel is
stopped.\n",
                           p));
                pChannelExitParams->ExitResponse  = MQXCC_CLOSE_CHANNEL;
                pChannelExitParams->ExitResponse2 = MQXR2_USE_AGENT_BUFFER;
            }

            p = (char*)strtok_r(NULL, "/", &strtok_cb);  /* look for next token */
        }

        /* NOTE: pExitBufferAddr is used to pass flags between exit points      */
        *pExitBufferAddr = (PMQPTR)lOptions;
        DBGPRINTF((OST,"itsoME: init resulting options %ld\n",lOptions));
        break;

    case MQXR_TERM:                                  /* Termination             */
        DBGPRINTF((OST,"itsoME: term\n"));
        DBGLOG((OSL,"itsoME: ver=%s ChannelName=\"%.*s\" ended.\n",
                VERSION,
                rlnts(MQ_CHANNEL_NAME_LENGTH, pChannelDefinition->ChannelName),
                pChannelDefinition->ChannelName));
        /* no action */
        break;

    case MQXR_MSG:                                   /* Message                 */
        /* NOTE: pExitBufferAddr is used to pass flags between exit points      */
        lOptions = (MQLONG)*pExitBufferAddr;
        DBGPRINTF((OST,"itsoME: msg MsgType=%d Report=0x%08x\n",
                  pXqh->MsgDesc.MsgType,
                  pXqh->MsgDesc.Report));

        if((pChannelDefinition->ChannelType == MQCHT_RECEIVER
                ||  pChannelDefinition->ChannelType == MQCHT_REQUESTER))
        {

            /* check if incoming msg is asking for exception/expiration report  */
            if ((lOptions & MQME_EXP)
                    && (pXqh->MsgDesc.Report & ( MQRO_EXCEPTION
                                               | MQRO_EXCEPTION_WITH_DATA
                                               | MQRO_EXCEPTION_WITH_FULL_DATA
                                               | MQRO_EXPIRATION
                                               | MQRO_EXPIRATION_WITH_DATA
                                               | MQRO_EXPIRATION_WITH_FULL_DATA)))
            {
                /* And if so remove the EXCEPTION/EXPIRATION options            */
                pXqh->MsgDesc.Report = pXqh->MsgDesc.Report
                                     & ( ~MQRO_EXCEPTION
                                       & ~MQRO_EXCEPTION_WITH_DATA
```

```
                                & ~MQRO_EXCEPTION_WITH_FULL_DATA
                                & ~MQRO_EXPIRATION
                                & ~MQRO_EXPIRATION_WITH_DATA
                                & ~MQRO_EXPIRATION_WITH_FULL_DATA);
        DBGPRINTF((OST,
            "itsoME: Unwanted Report options was removed (EXC/EXP).\n"));
    }

    /* check if incoming msg is asking for coa/cod report               */
    if((lOptions & MQME_COA)
            && (pXqh->MsgDesc.Report & ( MQRO_COA
                                       | MQRO_COA_WITH_DATA
                                       | MQRO_COA_WITH_FULL_DATA
                                       | MQRO_COD
                                       | MQRO_COD_WITH_DATA
                                       | MQRO_COD_WITH_FULL_DATA)))
    {
        /* And if so remove the COA/COD options                         */
        pXqh->MsgDesc.Report = pXqh->MsgDesc.Report
                             & ( ~MQRO_COA
                               & ~MQRO_COA_WITH_DATA
                               & ~MQRO_COA_WITH_FULL_DATA
                               & ~MQRO_COD
                               & ~MQRO_COD_WITH_DATA
                               & ~MQRO_COD_WITH_FULL_DATA);
        DBGPRINTF((OST,
             "itsoME: Unwanted Report options was removed (coa/cod).\n"));
    }

    /* check if incoming msg is ask us to discard the message in case   */
    /* of problems                                                      */
    if((lOptions & MQME_DISC)
            && (pXqh->MsgDesc.Report & ( MQRO_DISCARD_MSG
                                       | MQRO_PASS_DISCARD_AND_EXPIRY)))
    {
        /* And if so remove the COA/COD options */
        pXqh->MsgDesc.Report = pXqh->MsgDesc.Report
                             & ( ~MQRO_DISCARD_MSG
                               & ~MQRO_PASS_DISCARD_AND_EXPIRY);
        DBGPRINTF((OST,
             "itsoME: Unwanted Report options was removed (discard).\n"));
    }

    /* check if incoming msg enforce MCAUSER                            */
    if(lOptions & MQME_MCA)
    {
        memset(pXqh->MsgDesc.UserIdentifier,'\00',MQ_USER_ID_LENGTH);
        memcpy(pXqh->MsgDesc.UserIdentifier,
               pChannelDefinition->MCAUserIdentifier,
               MQ_USER_ID_LENGTH);
        DBGPRINTF((OST,"itsoME: Replaced UserIdentifier with \"%.*s\"\n",
                       MQ_USER_ID_LENGTH, pXqh->MsgDesc.UserIdentifier));
    }
}
```

```
        /* Modifying the ReplyToQmgr and ReplyToQ on normal channels to allow   */
        /* a flexible B2B integration without revealing your bussiness partners */
        /* when modifying your WebSphere MQ infrastructure.                     */
        /* Only change if ReplyToQ is in use                                    */
        if((pChannelDefinition->ChannelType == MQCHT_RECEIVER
                ||  pChannelDefinition->ChannelType == MQCHT_REQUESTER
                ||  pChannelDefinition->ChannelType == MQCHT_SENDER
                ||  pChannelDefinition->ChannelType == MQCHT_SERVER)
               &&(pXqh->MsgDesc.ReplyToQ[0]>' ')
               &&(lOptions & MQME_RTQM))
        {
/* prepare for changing ReplyToQmgr and ReplyToQ */
            memset(cExitDataWork,'\00',sizeof(cExitDataWork));
            memcpy(cExitDataWork, pChannelDefinition->MsgUserData,
                   MQ_EXIT_DATA_LENGTH);

            /* point to new ReplyToQmgr from MsgUserData                        */
            p = strstr(cExitDataWork,"RTQM=");
            if (p!=NULL)
            {
                p=p+5;
                /* Obtain the queue manager name                                */
                memset(cWorkRQmgr,'\00',sizeof(cWorkRQmgr));
                memset(cWorkRQmgr1,'\00',sizeof(cWorkRQmgr1));

                u=(char*)strtok(p,"/");
                if(u != NULL)  /* Should always be there                      */
                {
                    memcpy(cWorkRQmgr, p, rlnts(MQ_EXIT_DATA_LENGTH, p));
                    memcpy(cWorkRQmgr1, pXqh->MsgDesc.ReplyToQMgr,
                           rlnts(MQ_Q_MGR_NAME_LENGTH,pXqh->MsgDesc.ReplyToQMgr));
                }

                if (strcmp(cWorkRQmgr, cWorkRQmgr1))
                    {
                    DBGPRINTF((OST,"itsoME: Replaced WQmgr \"%s\" Qmgr1\"%s\"\n",
                               cWorkRQmgr, cWorkRQmgr1));

                    /* Build up the new ReplyToQueue                            */
                    memset(cWorkReplyQueue,'\00',sizeof(cWorkReplyQueue));
                    strncpy(cWorkReplyQueue,pXqh->MsgDesc.ReplyToQMgr,
                            rlnts(MQ_Q_MGR_NAME_LENGTH,
                            pXqh->MsgDesc.ReplyToQMgr));
                    strcat(cWorkReplyQueue,".");
                    strncat(cWorkReplyQueue, pXqh->MsgDesc.ReplyToQ,
                            rlnts(MQ_Q_NAME_LENGTH, pXqh->MsgDesc.ReplyToQ));

                    DBGPRINTF((OST,"itsoME: Old ReplyToQ \"%.*s\"\n",
                               rlnts(MQ_Q_MGR_NAME_LENGTH,
                               pXqh->MsgDesc.ReplyToQ),
                               pXqh->MsgDesc.ReplyToQ));

                    DBGPRINTF((OST,"itsoME: new ReplyToQ \"%s\"\n",
                               cWorkReplyQueue));
```

```
                    /*Adjust the ReplyToQ, just take MQ_Q_NAME_LENGTH          */
                    strncpy(pXqh->MsgDesc.ReplyToQ,cWorkReplyQueue,
                            rlnts(MQ_Q_NAME_LENGTH, cWorkReplyQueue));

                    /* And now change the replyToQMgr                          */
                    memset(pXqh->MsgDesc.ReplyToQMgr, '\00',
                           MQ_Q_MGR_NAME_LENGTH);
                    memcpy(pXqh->MsgDesc.ReplyToQMgr, cWorkRQmgr,
                           rlnts(MQ_Q_MGR_NAME_LENGTH, cWorkRQmgr));

                    DBGPRINTF((OST,
                               "itsoME: Replaced ReplyToQMgr with \"%.*s\"\n",
                               rlnts(MQ_Q_MGR_NAME_LENGTH,
                               pXqh->MsgDesc.ReplyToQMgr),
                               pXqh->MsgDesc.ReplyToQMgr));
                    DBGPRINTF((OST,
                               "itsoME: Replaced ReplyToQueue with \"%.*s\"\n",
                               rlnts(MQ_Q_NAME_LENGTH,pXqh->MsgDesc.ReplyToQ),
                               pXqh->MsgDesc.ReplyToQ));
                }
            }
        }

        break;                                      /* switch - Message        */

    default:                                        /* unsupported exit reason */
        DBGPRINTF((OST,"itsoME: error - unsupported exit reason\n"));
        /* prevent msg going any further */
        pChannelExitParams->ExitResponse = MQXCC_SUPPRESS_FUNCTION;
        break;

    }                                               /* switch - ExitReason     */

    DBGPRINTF((OST,"itsoME: exit ExitResponse=%s (%ld)\n\n",
               MQExitResponseType(pChannelExitParams),
               pChannelExitParams->ExitResponse));

    return;

}                                                   /* MsgExit                 */
/*****************************************************************************/
/*                              end                                          */
/*****************************************************************************/
```

B

Additional tooling for WebSphere MQ Internet pass-thru

This appendix discusses the scripts and exit that are used with WebSphere MQ Internet pass-thru in Chapter 10, “Scenario: Securing IBM WebSphere MQ connections to connect a business partner” on page 141. The scripts include the `init.d` script to start and stop WebSphere MQ Internet pass-thru, the WebSphere MQ Internet pass-thru control script, and a Java exit to implement IP address *whitelist* processing in WebSphere MQ Internet pass-thru routes.

This appendix contains the following topics:

- WebSphere MQ Internet pass-thru start/stop script
- WebSphere MQ Internet pass-thru control script
- WebSphere MQ Internet pass-thru security exit (itsoBlockExit)

WebSphere MQ Internet pass-thru start/stop script

The **mqipt.init.d** script is used to start and stop WebSphere MQ Internet pass-thru. The main function of the script is to validate the requesting user, and for the root user, switch to the mqm account before calling the **mqipt** control script to perform the actual work.

The **mqipt.init.d** script is designed to be copied into /etc/init.d/ and renamed to "**mqipt**". The Linux **chkconfig** utility is then used to configure the system for automatic start/stop processing of the script:

```
chkconfig --add mqipt
```

Once implemented, root or the mqm user can start or stop the mqipt instance using the standard Linux service interface:

```
service mqipt start
```

The stop processing attempts to use the CommandPort interface to WebSphere MQ Internet pass-thru, which is disabled for security reasons in the B2B scenario. The alternative option to shut down WebSphere MQ Internet pass-thru in this environment is the clean option:

```
service mqipt clean
```

Example B-1 shows the **mqipt.init.d** script. Changes that might be needed for specific sites are the user ID that you designate to run the mqipt process, the runlevels that **chkconfig** should activate, and the start/stop sequence numbers.

Example B-1 mqipt.init.d script

```
#!/bin/bash
#
# chkconfig: 35 99 01
# description: Starts/stops the MQIPT daemon

# This script is used to start up MQIPT daemon from
# linux startup (init.d).
#
# This script is owned by root:mqm with permission 754.
# The permission and ownership shouldn't be changed.
#
# LOCATION:     /etc/init.d/mqipt
#               Symbolic link from /etc/rc3.d/S99mqipt
#               to /etc/init.d/mqipt
#
# If MQIPT is controlled by VCS it should not be added to system startup using
# "chkconfig".
#

AWK=gawk;
ID=id;

BINDIR=/var/mqm/scripts
MQM=mqm
USER=`$ID -un`
ARG=$1
```

```
case $ARG in
  'start'|'stop'|'restart'|'status'|'monitor'|'clean'|'refresh')
        if [ "$USER" = "root" ] ; then
                su - $MQM -c "$BINDIR/mqipt $ARG"
        elif [ "$USER" = "mqm" ] ; then
                $BINDIR/mqipt $ARG
        else
                echo "You are not allowed to run mqipt."
                exit 1
        fi
        ;;


  *)
      echo "Usage: $0 { start | stop | restart | status | monitor | clean |
refresh }"
      exit 1
      ;;
esac
```

WebSphere MQ Internet pass-thru control script

The **mqipt** control script is designed to run from /var/mqm/scripts. It can be easily amended to run from a different location. The function of the script is to provide a simple and standardized interface to control a number of functions associated with WebSphere MQ Internet pass-thru.

The **mqipt** script performs these core functions:

- start: Start the mqipt daemon
- stop: Shut down the daemon using the control interface
- clear: Shut down the daemon using signals
- gui: Start the interactive configuration tool
- refresh: Use the control interface to tell the running daemon to refresh its configuration from the mqipt.conf file
- restart: Stop and restart the daemon

In addition, the script provides some inquiry-type functions that do not change the state of the running daemon:

- status: Show the current status of the daemon (running or down)
- monitor: Similar to status but returns a reason code for Veritas Cluster Server (VCS)
- jre: Show the Java Runtime Environment (JRE) and Java version that will be used by the mqipt daemon
- version: Show the version number of the **mqipt** script

Example B-2 shows the **mqipt** control script.

Example B-2 mqipt control script

```
#!/bin/ksh
#
#
#
# SCRIPT NAME:  mqipt
  VERSION="V1.0.3"
# LAST CHANGED: 10 July 2012
#
# LOCATION:     /var/mqm/scripts
#               Called from /etc/init.d/mqipt
#
# Method:       At system startup or manually as required.
# Usage:        mqipt { start | stop | status | monitor | clean | restart |
refresh | gui | jre }
#
# Description:  Starts / Stops WebSphere MQ Internet Passthrough daemon:
#
# Limitations:  This script is tested with MQIPT V2.0
#
# Depends on:   /var/mqm/mqipt
#
#-------------------------------------------------------------------------------
#
# Change History:  13 May 2011. V1.0.0
#                      Initial creation
#                  13 May 2011. V1.0.1
#                      Look for IBM Java 6 jvm in preference to
#                      Veritas VCS JVM, and then assume JVM in path
#                  10 July 2012. V1.0.3
#                      Look for MQ JRE before any other JRE.
#                      Add 'jre' option to display jre location
#
#-------------------------------------------------------------------------------
printversion()
{
        echo "mqipt script $VERSION"
}
printusage()
{
        printversion;
        echo "Usage: /var/mqm/scripts/mqipt { start | stop | clean | status |
monitor | restart | refresh | version | gui | jre }"
}
#-------------------------------------------------------------------------------
# Set Host name and type
#-------------------------------------------------------------------------------
THISHOST=`hostname`
IPTBIN=/opt/mqipt/bin
IPTDATA=/var/mqm/mqipt
IPTCMD="java -classpath /opt/mqipt/lib/com.ibm.mq.ipt.jar
com.ibm.mq.ipt.IPTController -D $IPTDATA"
KERNEL=`uname`
```

```
case "$KERNEL" in
'Linux')
        AWK=gawk;
        ID=id;
        if [ -x /opt/mqm/java/jre64/jre/bin/java ] ; then
                PATH=/opt/mqm/java/jre64/jre/bin:$PATH;
        elif [ -x /opt/mqm/java/jre/jre/bin/java ] ; then
                PATH=/opt/mqm/java/jre/jre/bin:$PATH;
        elif [ -x /opt/ibm/java-x86_64-60/jre/bin/java ] ; then
                PATH=/opt/ibm/java-x86_64-60/jre/bin:$PATH;
        elif [ -x /opt/VRTSjre/jre1.5/bin/java ] ; then
                PATH=/opt/VRTSjre/jre1.5/bin:$PATH;
        elif [ `which java | awk "{print \\$1}"` == no ] ; then
                echo "No Java found. Giving up"
                exit 1
        fi
        export PATH;
        ;;
'SunOS')
        AWK=nawk;
        ID=/usr/xpg4/bin/id;
        ;;
'AIX')
        AWK=nawk;
        ID=id;
        ;;
esac

#-----------------------------------------------------------------------------
# *************************      Start of Parameters        *******************
#-----------------------------------------------------------------------------

case "$1" in
#-----------------------------------------------------------------------------
# Print version info
#-----------------------------------------------------------------------------
'version' )
        printversion;
        ;;
#-----------------------------------------------------------------------------
# Print jre info
#-----------------------------------------------------------------------------
'jre' )
        echo "JRE used by MQIPT is" `which java`;
        java -version
        ;;
#-----------------------------------------------------------------------------
# Start the mqipt Server
#-----------------------------------------------------------------------------
'start' )
        cd $IPTDATA/logs
        nohup /opt/mqipt/bin/mqipt $IPTDATA &
        ;;
#-----------------------------------------------------------------------------
# Start the mqipt GUI for configuration
```

```
#-----------------------------------------------------------------------------
'gui' )
        cd /opt/mqipt/bin
        ./mqiptGui &
        ;;
#-----------------------------------------------------------------------------
# Stop the mqipt server
#-----------------------------------------------------------------------------
'stop' )
        /opt/mqipt/bin/mqiptAdmin -stop
        ;;
#-----------------------------------------------------------------------------
# Refresh the mqipt server (reload config)
#-----------------------------------------------------------------------------
'refresh' )
        /opt/mqipt/bin/mqiptAdmin -refresh
        ;;
#-----------------------------------------------------------------------------
# Kill the mqipt server
#-----------------------------------------------------------------------------
'clean' )
        pid=`ps -ef | grep -v grep | grep "$IPTCMD"|$AWK "{print \\$2}"`
        if [ "x$pid" != "x" ]
        then
                kill $pid;
                sleep 1;
        fi
        pid=`ps -ef | grep -v grep | grep "$IPTCMD"|$AWK "{print \\$2}"`
        if [ "x$pid" != "x" ]
        then
                kill -9 $pid;
        fi
        ;;
#-----------------------------------------------------------------------------
# Restart the mqipt server
#-----------------------------------------------------------------------------
'restart' )
        $0 stop;
        sleep 1;
        $0 clean;
        $0 start;
        ;;
#-----------------------------------------------------------------------------
# Print mqipt server status info
#-----------------------------------------------------------------------------
'status' | 'monitor' )
        pid=`ps -ef | grep -v grep | grep "$IPTCMD"|$AWK "{print \\$2}"`
        if [ "x$pid" != "x" ]
        then
            echo "mqipt ------   running with pid $pid"
            if [ $1 == 'monitor' ] ; then
                exit 110
            fi
        else
            echo "mqipt ------   down"
```

```
            if [ $1 == 'monitor' ] ; then
                exit 100
            fi
        fi
        ;;
# *************************        End of Parameters        *******************
#-----------------------------------------------------------------------------
* )
        printusage;
        ;;
esac
```

WebSphere MQ Internet pass-thru security exit (itsoBlockExit)

WebSphere MQ Internet pass-thru has a number of available security features to limit and validate incoming connections. One capability that is not available within the base product is the capability to implement a list of allowable connections. This would limit which source systems (by IP address) are allowed to connect to which of the TCP ports that WebSphere MQ Internet pass-thru is using.

The itsoBlockExit, which is shown in Example B-3, provides the ability to allow access to an incoming request based on the port number and the source IP address. It implements *whitelist* processing. That is, unless the request to connect to a port from that IP address appears in the list, the request will be denied.

Example B-3 itsoBlockExit that provides IP address control to WebSphere MQ Internet pass-thru

```
/*
 * Sample program for use with IBM WebSphere MQ Internet pass-thru
 * 5639-L92
 *
 * (C) COPYRIGHT International Business Machines Corp., 2003, 2012
 * All Rights Reserved * Licensed Materials - Property of IBM
 *
 * This sample program is provided AS IS and may be used, executed,
 * copied and modified without royalty payment by customer
 *
 * (a) for its own instruction and study,
 * (b) in order to develop applications designed to run with an IBM
 *     WebSphere product, either for customer's own internal use or for
 *     redistribution by customer, as part of such an application, in
 *     customer's own products.
 */

/*
 * compile command
 *
 * export CLASSPATH=/opt/mqipt/lib/com.ibm.mq.ipt.jar:.
 * javac -Xlint:unchecked itsoBlockExit.java
 *
 */

import java.io.BufferedReader;
import java.io.FileReader;
```

```
import java.util.Hashtable;

import com.ibm.mq.ipt.ConnectionListener;
import com.ibm.mq.ipt.Copyright;
import com.ibm.mq.ipt.exit.ExitRc;
import com.ibm.mq.ipt.exit.IPTTrace;
import com.ibm.mq.ipt.exit.SecurityExit;
import com.ibm.mq.ipt.exit.SecurityExitResponse;

/**
 * This is a sample  program to show how to use a WebSphere MQ
 * Internet pass-thru (MQIPT) security exit. It is provided as
 * a working example which will restrict route connection requests
 * based on the source address visible to MQIPT.
 * <p>
 * The destination port and source address are retrieved from the
 * structure passed to the program, and validated against
 * the content of the configuration file.
 * <p>
 * The sample config file (itsBlockExit.conf) contains the entries :
 *
 * <pre>
 *  5001 192.168.103.104
 *  6001 9.42.171.232
 * </pre>
 *
 * <p>
 * A security exit must extend {@link SecurityExit} to pickup
 * some predefined fields and perform some initialization before
 * it can be started. SecurityExit contains a set of predefined
 * methods which should be implemented in your security exit,
 * otherwise when the method is called by MQIPT it will only
 * perform a default action. This sample program has implemented
 * all recommended methods and shows how they can be used.
 * The methods are :
 * <ul>
 * <li>init - perform any initialization and make this exit
 *      ready to accept connections
 * <li>close - perform any cleanup of resources
 * <li>refresh - properties have been changed so we may need
 *      to reload control information
 * <li>validate - a connection request needs validating
 * </ul>
 * This sample also demonstrates how to use the tracing facility.
 * See the MQIPT book for more information on the trace facility.
 *
 * <pre>
 * History of changes :
 * Date     Author  Description
 * -------  ------  -----------
 * 10Feb03  PRB     Created
 * 12Mar03  PRB     Include sample refresh action
 * 16Jun08  PRB     Update to new interface for MQIPT v2
 * 09Jul12  NWC     Rework from SampleOneRouteExit to itsoBlockExit
 * </pre>
```

```
 *
 * @version 1.0
*/
public class itsoBlockExit extends SecurityExit {

  private static final String COPYRIGHT_NOTICE = Copyright.SHORT_STRING;
  /**
   * My class name
   */
  private static final String CLASS_NAME = "itsoBlockExit"; //$NON-NLS-1$
  /**
   * Config file name
   */
  private static final String CONFIG_FILE_NAME = CLASS_NAME + ".conf";
                                                            //$NON-NLS-1$
  /**
   * Fully qualified name of config file
   */
  private static final String CONFIG_FILE = getExitsDir() + CONFIG_FILE_NAME;
  /**
   * A table of IP addresses and port numbers. As this table is not
   * shared with any other instances of this class, using a Hashtable
   * is bit of an overkill (Hashtable uses synchronized methods), but
   * this table could easily be used for other purposes. The combined
   * String value of the string representation of the source IP
   * address, ":" and the string representation of the port number
   * without leading zeroes is used as the key. The value stored
   * against the key is the constant "valid". The existence of the
   * key in the hash table is used to indicate that the route is valid.
   */
  private Hashtable ipTable = new Hashtable();

  /**
   * Default constructor used by {@link ConnectionListener} which
   * creates a new instance of this class. No parameters are allowed
   * on this constructor as this class is loaded dynamically at
   * runtime and is created by Class.newInstance().
   */
  public itsoBlockExit() {

  }

  /**
   * This method is called when a route is being started and should
   * be used to perform any initialization (e.g. reading configuration
   * data) and place itself in a ready state to validate connection
   * requests. Read the configuration table and initialize the table
   * with a list of Queue Manager names and server names.
   *
   * @param t is a trace context
   * @return a reason code of {@link SecurityExitResponse#OK} or
   *         {@link SecurityExitResponse#INIT_ERROR}
   */
  public int init(IPTTrace t) {
    // Function id
```

```
  final String fid = "itsoBlockExit.init"; //$NON-NLS-1$

  // Reason code
  int rc = ExitRc.OK;

  // Trace entry
  t.entry(fid);

  // Trace useful info
  t.data(fid, "Starting security exit - MQIPT version " + getVersion());
                                                  //$NON-NLS-1$

  // Read config file and initialize hash table
  if (!loadTable(t)) {

    // Error
    rc = ExitRc.INIT_ERROR;
  }

  // We're ready to validate requests
  if (rc == ExitRc.OK) {
    t.data(fid, "Ready for work"); //$NON-NLS-1$
  }

  // Trace exit
  t.exit(fid);

  return rc;
}

/**
 * This method is called when the route is about to stop. It should
 * perform any cleanup operations.
 *
 * @param t is a trace context
 */
public void close(IPTTrace t) {
  // Function id
  final String fid = "itsoBlockExit.close"; //$NON-NLS-1$

  // Trace entry
  t.entry(fid);

  // Perform any cleanup
  ipTable.clear();

  // Trace exit
  t.exit(fid);
}

/**
 * This method is called to when the route has been refreshed. It will
 * clear out the current data in the table and reload new data
 * from the configuration file.
 *
```

```
 * @param t is a trace context
 * @return {@link SecurityExitResponse#OK} or
 * {@link SecurityExitResponse#REFRESH_ERROR}
 */
public int refresh(IPTTrace t) {
  // Function id
  final String fid = "itsoBlockExit.refresh"; //$NON-NLS-1$

  // Reason code
  int rc = ExitRc.OK;

  // Trace entry
  t.entry(fid);

  // Empty the old table
  t.data(fid, "Clearing out old table"); //$NON-NLS-1$

  ipTable.clear();

  // Read config file and initialize hash table
  if (!loadTable(t)) {
    t.data(fid, "Error refreshing table"); //$NON-NLS-1$

    // Trace exit
    rc = ExitRc.REFRESH_ERROR;
  }

  // Trace exit
  t.exit(fid);

  return rc;
}

/**
 * This method is called to validate a connection request. It will
 * build a search argument from the ListenerPort and the
 * ClientIPAddress, and query the table (ipTable) built during
 * initialization.
 *
 * @param t is a trace context
 * @return {@link SecurityExitResponse} containing the response
 */
public SecurityExitResponse validate(IPTTrace t) {
  // Function id
  final String fid = "itsoBlockExit.validate"; //$NON-NLS-1$

  SecurityExitResponse secExitResponse = null;

  // Trace entry
  t.entry(fid);

  // Trace input parameters
  t.data(fid, listParameters());

  // Get WMQ ClientIPAddress
```

```
    String clientIPAddress = getClientIPAddress();

    // Get WMQ ListenerPort
    Integer listenerPort = getListenerPort();

    // Convert Listener Port number to String
    String listenerPortString = listenerPort.toString();

    // Build composite value for key
    String thisRequest = clientIPAddress.concat(":").concat(listenerPortString);

    // Write a trace showing the lookup we need to match
      t.data(fid, "Looking up : " + thisRequest); //$NON-NLS-1$

    // Check we know about the specified IP Address and Port
    if (isValidRequest(t, thisRequest)) {

      // Allow the connection request
      secExitResponse = new SecurityExitResponse(ExitRc.OK);

      // Trace result
      t.data(fid, "Connection allowed\n" + secExitResponse.toString());
                                                //$NON-NLS-1$
    }
    // We don't know about the IP Address and Port combination
    else {
      // Reject the connection request
      secExitResponse = new SecurityExitResponse(ExitRc.NOT_AUTHORIZED);

      // Trace result
      t.data(fid, "Connection not allowed - source not valid for port");
                                                //$NON-NLS-1$
    }

    // Trace exit
    t.exit(fid);

    return secExitResponse;
  }

  /**
   * Read the config file and initialize a list of IP source addresses
   * and dest ports. The config file is read and a HashTable of values
   * is built, combining IP address and dest port, and storing a
   * constant in the hash.
   * The format of each line in the config file must be : <Port> <IP address>
   *
   * @param t is a trace context
   * @return <code>true</code> if list successfully created or
   * <code>false</code> if any errors
   */
  private boolean loadTable(IPTTrace t) {
    // Function id
    final String fid = "itsoBlockExit.loadTable"; //$NON-NLS-1$
```

```
// Input config file
BufferedReader br = null;

// Line number in config file
int lines = 0;

// Line of text from config file
String text = null;

// Trace entry
t.entry(fid);

try {
  // Trace config file name
  t.data(fid, "Opening file " + CONFIG_FILE); //$NON-NLS-1$

  // Open config file
  br = new BufferedReader(new FileReader(CONFIG_FILE));

  // Read the config file - one line at a time
  while ((text = br.readLine()) != null) {

    // Ignore any lines starting with a "#" - a comment line
    if (text.startsWith("#")) //$NON-NLS-1$
      continue;

    t.data(fid, "Found line : " + text); //$NON-NLS-1$

    // Split line in 2 - 1st part is Dest Port
    //                 - 2nd part is source IP address
    int i = text.indexOf(" "); //$NON-NLS-1$
    if (i < 0) {
      t.data(fid, "Parsing error, line " + lines); //$NON-NLS-1$

      // Trace exit
      t.exit(fid);

      // Signal an error
      return false;
    }

    // Destination Port
    String destPort = text.substring(0, i).trim();
    // Source IP address
    String sourceIP = text.substring(i + 1).trim();

    // Build composite value for key
    String validRequest = sourceIP.concat(":").concat(destPort);

    // Add entry to list
    addToTable(t, validRequest, (String) "valid");

    // Increment line number
    lines++;
  }
```

```
      // Finished with config file
      br.close();

      t.data(fid, "Closing file " + CONFIG_FILE); //$NON-NLS-1$
    }
    catch (Exception e) {
      t.data(fid, "Exception loading table\n" + e.getMessage()); //$NON-NLS-1$

      try {
        br.close();
      }
      catch (Exception e2) {}

      // Trace exit
      t.exit(fid);

      // Signal an error
      return false;
    }

    // Trace exit
    t.exit(fid);

    // All done successfully
    return true;
  }

  /**
   * Add an entry to the table. An entry consists of a source IP
   * address concatenated to ":" and the dest Port. The value
   * added for each key is never used.
   *
   * @param t is a trace context
   * @param validRequest is the <source IP>:<dest port>
   * @param dummy is any String
   */
  private void addToTable(IPTTrace t, String validRequest, String dummy) {
    // Function id
    final String fid = "itsoBlockExit.addToTable"; //$NON-NLS-1$

    // Trace entry
    t.entry(fid);

    // Use Queue Manager name as the key
    ipTable.put(validRequest, dummy);

    t.data(fid, "Added " + validRequest + " to table"); //$NON-NLS-1$
                                                        //$NON-NLS-2$

    // Trace exit
    t.exit(fid);

    return;
  }
```

```
  /**
   * Determine if the source_address:dest_port combination is valid.
   * If it is found in the table, return  <code>true</code>. If the
   * value is not found, return <code>false</code>.
   *
   * @param t is a trace context
   * @param channelName
   * @return <code>true</code> if the request combination is found.
   *         <code>false</code> if the combination is not found.
   *         found
   */
  private boolean isValidRequest(IPTTrace t, String ipRequest) {
    // Function id
    final String fid = "itsoBlockExit.isValidRequest"; //$NON-NLS-1$

    // Trace entry
    t.entry(fid);

    // Check if table is empty - this should never really happen
    if (ipTable.isEmpty()) {
      t.data(fid, "Table is empty"); //$NON-NLS-1$

      // Trace exit
      t.exit(fid);

      return false;
    }

    t.data(fid, "Looking for ip Address and port : " + ipRequest);
                                                  //$NON-NLS-1$

    // Get the dummy string
    String dummyValue = (String) ipTable.get(ipRequest);

    // Check we have a server name
    if (dummyValue == null) {
      t.data(fid, "IP address/port not in table"); //$NON-NLS-1$

      // Trace exit
      t.exit(fid);

      return false;
    }

    t.data(fid, "Found IP address and port : " + ipRequest); //$NON-NLS-1$

    // Trace exit
    t.exit(fid);

    return true;
  }
}
```

C

Certificate administration techniques and special WebSphere MQ security checks

This appendix contains additional information about administering certificates on z/OS. It includes snippets of Job Control Language (JCL) that are designed to help administrators with Secure Sockets Layer (SSL)/Transport Layer Security (TLS) certificate administration using IBM Resource Access Control Facility (RACF). In addition, it provides a sample REXX executable that can help you check for some common yet critical IBM WebSphere MQ security issues.

This appendix contains the following topics:

- Managing certificates on z/OS
- Simple z/OS MQ security check

Managing certificates on z/OS

SSL/TLS certificates on z/OS are managed using **RACDCERT** commands, whose name derives from the fact they are used to manage RACF digital certificates. Additional information about these commands is available from the following website:

- *z/OS V1R12.0 Security Server RACF Command Language Reference*:

  http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=%2Fcom.ibm.zos.r12.icha400%2Fichza4b0.htm

The information that is presented here expands on the topics that are first discussed in 8.11, “WebSphere MQ (CMS) keystores” on page 92.

Generating a certificate and key

Use the **GENCERT** command to create a new private self-signed certificate and a new private key. Example C-1 shows the command sequence with the key length set to 2048 bits.

Example C-1 JCL to generate new private certificate (self-signed)

```
//GENCERT JOB
//*******************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- Create the servers self-signed certificate
//*-- MQs personal certificate
//*-----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
PROF MSGID
RACDCERT ID(STC) GENCERT +
    SUBJECTSDN( +
    CN('CSQ1') +
    OU('SA-W216 Residency') +
    O('IBM ITSO') +
    C('US')) +
    SIZE(2048) +
    WITHLABEL('ibmWebSphereMQCSQ1')
/*
//
```

Example C-2 shows the output of the **GENCERT** command sequence from the previous example.

Example C-2 Output of the create-a-new-certificate sequence

```
RACDCERT ID(STC) GENCERT SUBJECTSDN( CN('CSQ1') OU('SA-W216 Residency') O('IBM
ITSO') C('US')) SIZE(2048) WITHLABEL('ibmWebSphereMQCSQ1')
IRRD175I The new profile for DIGTCERT will not be in effect until a SETROPTS
REFRESH has been issued.
READY
```

Creating a certificate request

The **GENREQ** command generates a certificate request for signing the certificate, as shown in Example C-3. This will create a base64-encoded file, so that you can just pass the content of the CERTREQ file on to the certificate authority (CA). This command is also used for renewing your CA certificate.

Example C-3 JCL to generate a certificate request for signing by the CA

```
//GENCRQST JOB
//*****************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*----------------------------------------------------------------
//*-- Create a CSR in base64-encoded text
//*-- Can be used for a renewal request OR
//*-- Can be used as step 2 when setting up a server
//*-- Job GENCERT is step 1 in that case
//*-- When the CA returns the signed cert run job INSTCERT
//*----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT GENREQ (LABEL('ibmWebSphereMQCSQ1'))  ID(STC) +
  DSN('JOPDE.CSQ1.CERTREQ')
/*
//
```

Installing a certificate

The **ADD** command is used to update a private certificate with the CA-signed parts, as shown in Example C-4. The private key stays safe and is not changed. Only the certificate is updated with the information that is received from the CA.

Example C-4 JCL to install a signed certificate

```
//INSTCERT JOB
//*****************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*----------------------------------------------------------------
//*-- Add the signed CSR and replace the existing certificate
//*----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT ADD('JOPDE.CSQ1CERT.CERV') ID(STC)
/*
//
```

Installing CA certificates in the truststore

This job (see Example C-5) uses the **ADD** command to install CA certificates in the RACF truststore. The truststore contains CA certificates only.

Example C-5 JCL to add CA certificates to a truststore

```
//INSTCA JOB
//*****************************************************************
```

```
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- Add a certificate authority (CA)  certificate to the truststore
//*-----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT CERTAUTH    +
    ADD('JOPDE.ROOTCA.DERBIN') +
    WITHLABEL('WMQCA1 MQ Root CA')
RACDCERT CERTAUTH    +
    ADD('JOPDE.IMMEDCA.DERBIN') +
    WITHLABEL('WMQCA2 MQ Intermediate CA')
/*
//
```

Creating a keyring

Example C-6 shows the job for creating a keyring by using the **ADDRING** command. A keyring is used to hold keys that are used for a certain application.

Example C-6 JCL to create a keyring to which certificates can be connected

```
//ADDKRING JOB
//*****************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- Create keyring and add it to a user
//*-----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT ID (STC) ADDRING (IBM1RING)
/*
//
```

Connecting certificates to a keyring

Certificates are connected to the keyring by using the **CONNECT** command that is shown in Example C-7. Applications, such as WebSphere MQ, can only access and utilize certificates that are in the keyring to which it is connected.

Example C-7 JCL to connect certificates to a keyring

```
//CONRING JOB
//*****************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- Connect root and private certificates to KEYRING IBM1RING
//*-----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
PROF MSGID
RACDCERT ID(STC) CONNECT (CERTAUTH  +
    LABEL('WMQCA1 MQ Root CA')    +
    RING(IBM1RING) +
```

```
    USAGE (CERTAUTH))
RACDCERT ID(STC) CONNECT (CERTAUTH  +
    LABEL('WMQCA2 MQ Intermediate CA') +
    RING(IBM1RING) +
    USAGE (CERTAUTH))
RACDCERT ID(STC) CONNECT (ID(STC) +
    LABEL('ibmWebSphereMQCSQ1') +
    RING(IBM1RING) +
    USAGE(PERSONAL))
/*
//
```

Deleting a certificate

You can use the **DELETE** command to delete a certificate and its keys from RACF, as shown in Example C-8.

> **Important:** The act of deleting a certificate cannot be reversed. A deleted certificate would need to be re-created by using the same steps that were used originally.

Example C-8 Command sequence to delete a certificate

```
//DELCERT JOB
//*****************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*----------------------------------------------------------------
//*-- Delete an existing certificate
//*----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
RACDCERT DELETE (LABEL('ibmWebSphereMQCSQ1'))  ID(STC)
/*
//
```

Example C-9 shows the output from the **DELETE** command sequence in the previous example.

Example C-9 Output of the delete certificate process

```
RACDCERT DELETE (LABEL('ibmWebSphereMQCSQ1'))  ID(STC)
RACLISTed profiles for DIGTCERT will not reflect the deletion(s) until a SETROPTS
REFRESH is issued.
```

Listing a certificate

Use the **LIST** command to see the contents of a certificate, as shown in Example C-10. The output includes information, such as the certificate's start date, end date, and current status.

The Status result is particularly informative. If the certification path is incomplete, it shows NOTRUST (see Example C-12 on page 312).

Example C-10 JCL to list a certificate

```
//LISTSERT JOB
//*****************************************************************
```

```
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- List existing certificates
//*-----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
PROF MSGID
RACDCERT ID(STC) list(LABEL('ibmWebSphereMQCSQ1'))
/*
//
```

Example C-11 shows the output of the **LIST** command where the certification path is complete (the Status field shows TRUST).

Example C-11 List certificate output where the certification path is complete

```
RACDCERT ID(STC) list(LABEL('ibmWebSphereMQCSQ1'))

Digital certificate information for user STC:

  Label: ibmWebSphereMQCSQ1
  Certificate ID: 2QPi48OJgpTmhYLil4iFmYXU2MPi2PFA
  Status: TRUST
  Start Date: 2012/07/10 00:00:00
  End Date:   2013/07/10 23:59:59
  Serial Number:
       >00<
  Issuer's Name:
       >CN=CSQ1.OU=SA-W216 Residency.O=IBM ITSO.C=US<
  Subject's Name:
       >CN=CSQ1.OU=SA-W216 Residency.O=IBM ITSO.C=US<
  Key Type: RSA
  Key Size: 2048
  Private Key: YES
  Ring Associations:
  *** No rings associated ***
```

Example C-12 shows the output of the **LIST** command for a certificate where the certification path is incomplete. It shows that the certificate is not trusted (NOTRUST). To fix this, you would need to install the missing certificates and add the signed certificate again.

Example C-12 List certificate output where the certification path is incomplete

```
RACDCERT ID(STC) list(LABEL('ibmWebSphereMQCSQ1'))

Digital certificate information for user STC:

  Label: ibmWebSphereMQCSQ1
  Certificate ID: 2QPi48OJgpTmhYLil4iFmYXU2MPi2PFA
  Status: NOTRUST
  Start Date: 2012/07/10 23:11:01
  End Date:   2013/06/30 12:36:05
  Serial Number:
       >35E6242D00000000002F<
  Issuer's Name:
       >CN=WMQCA2 MQ Intermediate CA.OU=SA-W216 Residency.O=IBM ITSO.C=US<
```

```
  Subject's Name:
       >CN=CSQ1.OU=SA-W216 Residency.O=IBM ITSO.C=US<
  Key Type: RSA
  Key Size: 2048
  Private Key: YES
  Ring Associations:
  *** No rings associated ***
```

Listing a keyring

The **LISTRING** command (refer to Example C-13) shows the content of a keyring (that is, for the certificates that WebSphere MQ knows about).

Example C-13 JCL to list the content of a keyring

```
//LSTKRING JOB
//*******************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- List certificates connected to a keyring
//*-----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
PROF MSGID
RACDCERT ID(STC) LISTRING(IBM1RING)
/*
//
```

Example C-14 shows the output of the **LISTRING** command sequence from the previous example.

Example C-14 Output of the list keyring sequence

```
RACDCERT ID(STC) LISTRING(IBM1RING)

Digital ring information for user STC:

  Ring:
       >IBM1RING<
  Certificate Label Name             Cert Owner     USAGE      DEFAULT
  -------------------------------    ------------   --------   -------
  WMQCA1 MQ Root CA                  CERTAUTH       CERTAUTH     NO

  WMQCA2 MQ Intermediate CA          CERTAUTH       CERTAUTH     NO

  ibmWebSphereMQCSQ1                 ID(STC)        PERSONAL     NO
```

Changing a certificate label

Use the **ALTER** command (see Example C-15 on page 314) to change the certificate label. This procedure can be useful in cases where there was a typographical error in the original label, or when activating a queue sharing group (QSG) that should take over the certificate from the main queue manager in the QSG. See 7.1.7, “Certificate sharing in a queue sharing group” on page 78.

Example C-15 JCL to change a certificate label

```
//RENCERT JOB
//*****************************************************************
//IKJEFT01 EXEC PGM=IKJEFT01
//*----------------------------------------------------------------
//*-- Change certificate label.
//*----------------------------------------------------------------
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
PROF MSGID
RACDCERT ALTER ( LABEL('ibmWebSphereMQIBM1')) +
    ID(STC) +
    NEWLABEL('ibmWebSphereMQCSQ2')
//*
```

Simple z/OS MQ security check

The simple z/OS security check programs that are provided in this section are intended to inspire you to be thorough and innovative in maintaining your secure WebSphere MQ installation on z/OS.

These programs aim to catch hidden or unusual security flaws that might otherwise go unnoticed, and to perform some basic inbound channel security checks. The controls are based on the authors’ years of experience working with WebSphere MQ.

Important: To execute these programs, you need high access rights to RACF, which could require special authorization from the IT department at your organization.

If executed with sufficient RACF authority, these programs perform the following checks:

- Security is enabled
- RESLEVEL security is not granted CONTROL or ALTER
- Universal access is not granted to SYSTEM command queues
- Universal access is not granted to PUB/SUB base TOPIC
- Mixed-case migration is enabled
- Channel authentication is enabled
- SSL is enabled on queue managers
- Inbound SYSTEM.* channels are locked down
- Inbound channels are protected with SSL, Message Channel Agent (MCA), and so on

MQCHECK performs checks based on the authority that is held by the user who executes the program. Install and run MQCHECK to get an indication of the security on your system.

The program started as a way to check for WebSphere MQ channel initiator (CHIN) security vulnerabilities that can result from giving too much authority to the CHIN, which causes WebSphere MQ to turn off security, completely, for the connection to the CHIN. In such situations, even invalid users, such as DUMMY, Nobody, or +:;%&¤#, would be accepted.

MQCHECK checks that the migration to mixed case security has been performed. Mixed case security was introduced with WebSphere MQ Version 7.0 to increase security by protecting lowercase WebSphere MQ objects together with Publish/Subscribe (PUB/SUB) functionality. PUB/SUB is only protected when mixed case security is enabled.

MQCHECK also checks that inbound channels have security enabled according to the basic hardening that you learned about in 6.4, “Channels, transmission queues, and communications” on page 56.

MQCHECK is not considered as a complete security control program, where all aspects of WebSphere MQ security are validated. It is intended to be a starting point, and should be adapted according to your company’s security policy needs.

Example C-16 shows the JCL to execute the **MQCHECK** script against one queue manager.

Example C-16 JCL to execute the REXX script MQCHECK on one queue manager (CSQ1)

```
//MQCHECK JOB
//*******************************************************************
//*-----------------------------------------------------------------
//*-- Obtain QMGR and CHANNEL definitions from queue manager CSQ1
//*-----------------------------------------------------------------
//*
//COMMAND EXEC PGM=CSQUTIL,PARM='CSQ1'
//STEPLIB  DD  DISP=SHR,DSN=MQ710.SCSQANLE
//         DD  DISP=SHR,DSN=MQ710.SCSQAUTH
//OUTPUT1  DD  DSN=&&QMGR,DISP=(MOD,PASS),UNIT=VIO,SPACE=(TRK,(7,7))
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
* THE NEXT COMMAND WILL COME FROM 'MQCMD' DDNAME
COMMAND DDNAME(MQCMD) MAKEDEF(OUTPUT1)
/*
//* Obtain QMGR and CHANNEL definitions
//MQCMD    DD *
DISPLAY QMGR ALL
DISPLAY chl(*) ALL
/*
//*
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- execute MQCHECK program to perform security check on the
//*-- specified WebSphere MQ subsystems on current z/OS
//*-- analyzing QMGR and Channel definitions
//*-----------------------------------------------------------------
//SYSEXEC  DD  DISP=SHR,DSN=CSQ1V71.SCSQPROC
//SYSTSPRT DD  SYSOUT=*
//INDD1    DD  DISP=(MOD,PASS),DSN=&&QMGR
//SYSTSIN  DD  *
 %MQCHECK CSQ1-INDD1
/*
//
```

Example C-17 shows the JCL to execute MQCHECK on RACF for defined WebSphere MQ subsystems on the logical partition (LPAR) where the job executes.

Example C-17 JCL to execute MQCHECK to check RACF settings for all queue managers

```
//MQCHECK JOB
//IKJEFT01 EXEC PGM=IKJEFT01
//*-----------------------------------------------------------------
//*-- execute MQCHECK program to perform security check on known
//*-- WebSphere MQ subsystems on current z/OS
//*-----------------------------------------------------------------
//SYSEXEC  DD  DISP=SHR,DSN=CSQ1V71.SCSQPROC
//SYSTSPRT DD  SYSOUT=*
//SYSTSIN  DD  *
 %MQCHECK
/*
//
```

Examples of MQCHECK results

The following examples of MQCHECK reports should encourage you to suggest better security requirements to the WebSphere MQ administrators.

Example C-18 shows the result of an MQCHECK that is run against a single queue manager that is insufficiently secured.

Example C-18 Output from MQCHECK when system has not been properly secured

```
****************************** MQCHECK v1r0m0 ******************************
Executed on: SC61 by JOPDE date: 16 Jul 2012 20:04:31 Opt: CSQ1-INDD1
****************************************************************************

This program is based AS-IS and there are no guarantees that it finds all security
holes, breaches. This solely is intended to display issues about CHIN task
issues and on moving from uppercase classes (MQ...) mixed case classes (MX...)

A user of this program should understand that IBM cannot provide technical
technical support for the program and will not be responsible for any
consequences of use of the program.

Regards
WebSphere MQ security Redbooks team 2012

----------------------------------------------------------------------------
Statistics from class query performed on: SC61

              MXADMIN  MQADMIN  MXQUEUE  MQQUEUE  MXTOPIC   MQCMDS
                   12       92        4      122        6      166
PUBSUB is enabled, PUBSUB Security need a check.

Validating queue manager: CSQ1
  CSQ1CHIN run under STC
    User STC is connected to SYS1 TSO SAPR3 AOPADMIN DB2PM CICS STORADM BANKDOS
    BANKUNO DNISECGR MQMSUPP WASCFG
  Queue manager: CSQ1 MQADMIN ** (G) UACC(ALTER)

----------------------------------------------------------------------------
```

```
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
Channel security is turned off due to universal access to MQADMIN ** (G)

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
There might be security issues with class MQCMDS  for  CSQ*.** (G)
located from: ALTER.QMGR UACC(ALTER)

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
There might be security issues with class MQQUEUE  for  ** (G)
located from: SYSTEM.COMMAND.QUEUE UACC(ALTER)

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
There might be security issues with class MQQUEUE  for  ** (G)
located from: SYSTEM.ADMIN.COMMAND.QUEUE UACC(ALTER)

SSL/TLS keyring settings seem to be fine.
Channel authentication is enabled on queue manager CSQ1
  rules should be validated.
-------------------------------------------------------------------------------

Channel security report:

Channel: SVRCONN.W.SCYEXIT assure that security exit: PWEXIT7
  perform real validation, to prevent unauthorized usage.

Channel: SYSTEM.ADMIN.SVRCONN might not be sufficient secured,
  check that MCAUSER('JOPDE') have no administrative rights.

Channel: SYSTEM.DEF.CLUSRCVR should be disabled to prevent unauth. usage.
  MCAUSER() and/or SCYEXIT() is blank or MAXMSGL > 1

Channel: SYSTEM.DEF.RECEIVER should be disabled to prevent unauth. usage.
  MCAUSER() and/or SCYEXIT() is blank or MAXMSGL > 1

Channel: SYSTEM.DEF.REQUESTER should be disabled to prevent unauth. usage.
  MCAUSER() and/or SCYEXIT() is blank or MAXMSGL > 1

Channel: SYSTEM.DEF.SVRCONN should be disabled to prevent unauth. usage.
  MCAUSER() and/or SCYEXIT() is blank or MAXMSGL > 1

Channel: WAS.JMS.SVRCONN is not protected according to best practice,
  MCAUSER(), SCYEXIT() and SSLCIPH() are all blank
READY
```

Example C-19 shows the result from MQCHECK where administrators have not migrated to use mixed case (MX) classes or disabled the PUB/SUB function.

Example C-19 Output of MQCHECK for system where PUB/SUB security has been enabled

```
****************************** MQCHECK v1r0m0 ******************************
Executed on: SC61 by JOPDE date: 16 Jul 2012 20:07:55 Opt: CSQ1-INDD1
*****************************************************************************
```

```
This program is based AS-IS and there is no guarantee that it finds all security
holes, breaches. This solely is intended to display issues about CHIN task
issues and on moving from uppercase classes (MQ...) mixed case classes (MX...)

A user of this program should understand that IBM cannot provide technical
technical support for the program and will not be responsible for any
consequences of use of the program.

Regards
WebSphere MQ security Redbooks team 2012

-------------------------------------------------------------------------------
Statistics from class query performed on: SC61

                MXADMIN  MQADMIN  MXQUEUE  MQQUEUE  MXTOPIC   MQCMDS
                     12       92        4      122        6      166
PUBSUB is enabled, PUBSUB Security need a check.

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
MXTOPIC profiles are is NOT used. PUBSUB is not protected.
MXTOPIC profiles are is NOT used. PUBSUB is not protected.
MXTOPIC profiles are is NOT used. PUBSUB is not protected.
MXTOPIC profiles are is NOT used. PUBSUB is not protected.
MXTOPIC profiles are is NOT used. PUBSUB is not protected.
*******************************************************************************

Validating queue manager: CSQ1
  CSQ1CHIN run under STC
    User STC is connected to SYS1 TSO SAPR3 AOPADMIN DB2PM CICS STORADM BANKDOS
    BANKUNO DNISECGR MQMSUPP WASCFG
  Queue manager: CSQ1 MQADMIN ** (G) UACC(ALTER)
.....
```

Example C-20 shows the result of an MQCHECK that was performed by a WebSphere MQ administrator without SPECIAL/AUDIT authority. SSL/TLS is not enabled.

Example C-20 MQCHECK output of WebSphere MQ V7.0.1 system without SPECIAL/AUDIT authority

```
****************************** MQCHECK v1r0m0 ******************************
Executed on: SC62 by JOPDE date: 16 Jul 2012 21:48:40
*******************************************************************************
......
-------------------------------------------------------------------------------
Statistics from class query performed on: SC62

                MXADMIN  MQADMIN  MXQUEUE  MQQUEUE  MXTOPIC   MQCMDS
                     -1       62       -1     1214       -1      137
The following RACF classes are not defined:
  MXQUEUE MXADMIN MXTOPIC MXPROC MXNLIST

PUBSUB is disabled on this queue manager
  No MX class checks are performed.

Validating queue manager: CSQ6
  ICH13002I NOT AUTHORIZED TO LIST **
```

```
  IRR52021I You are not authorized to view STDATA segments.
  Check not performed.

*** MQCHECK should be executed by a security administrator

SSL/TLS are not used. Authentication should be considered.
See Infocenter for additional information
-------------------------------------------------------------------------------

Channel security report:
...............
```

Example C-21 shows the result of an MQCHECK that was run against a queue manager where WebSphere MQ administrators have chosen to switch off security.

Example C-21 Output of MQCHECK when NO.SUBSYS.SECURITY has been defined

```
****************************** MQCHECK v1r0m0 ******************************
Executed on: SC61 by JOPDE date: 16 Jul 2012 20:16:57 Opt:
*******************************************************************************
....
Validating queue manager: MQA1

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
There might be security issues with class MQADMIN  for  MQA1.NO.SUBSYS.SECURITY
Located from: NO.SUBSYS.SECURITY

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
ALL SECURITY IS SWITCHED OFF. NO FURTHER CHECKS MADE MQA1
ALL SECURITY IS SWITCHED OFF. NO FURTHER CHECKS MADE MQA1
ALL SECURITY IS SWITCHED OFF. NO FURTHER CHECKS MADE MQA1
ALL SECURITY IS SWITCHED OFF. NO FURTHER CHECKS MADE MQA1
ALL SECURITY IS SWITCHED OFF. NO FURTHER CHECKS MADE MQA1
....
```

Example C-22 shows the result of an MQCHECK that was performed against a version 6 queue manager, without SSL/TLS enablement.

Example C-22 MQCHECK output when executed against a WebSphere MQ Version 6 queue manager

```
.....
-------------------------------------------------------------------------------
Statistics from class query performed on: ZOS1

                MXADMIN  MQADMIN  MXQUEUE  MQQUEUE  MXTOPIC   MQCMDS
                     -1        7       -1       12       -1        2
Not a version 7 queue manager.
  No MX class checks are performed.

Validating queue manager: CSQ6
  CSQ6CHIN run under START2
    User START2 is connected to SYS1 MQADM
  Queue manager: CSQ6 MQADMIN CSQ%.RESLEVEL (G) UACC(NONE) seems to be fine
```

```
  for CHIN security. Recommended setting.

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
There might be security issues with class MQADMIN  for  *.NO.CMD.CHECKS (G)
Located from: NO.CMD.CHECKS
RACF Class: MXQUEUE is not defined.
RACF Class: MXQUEUE is not defined.
RACF Class: MXTOPIC is not defined.
RACF Class: MXTOPIC is not defined.

SSL/TLS are not used. Authentication should be considered.
See Infocenter for additional information
-------------------------------------------------------------------------------
....
```

Example C-23 shows the result of an MQCHECK that was run where some users have security checks disabled.

Example C-23 Example where some users have no security

```
....
Validating queue manager: CSQ2
  CSQ2CHIN run under STC
    User STC is connected to SYS1 TSO SAPR3 AOPADMIN DB2PM CICS STORADM BANKDOS
    BANKUNO DNISECGR MQMSUPP WASCFG
  Users/groups connected to CSQ2.RESLEVEL : MQW1GRP
  Queue manager: CSQ2 MQADMIN CSQ2.RESLEVEL  UACC(NONE) seems to be fine
  for CHIN security. Recommended setting.

-------------------------------------------------------------------------------
WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING WARNING
The following users have security checks disabled:
  MQW1 MQW1BRK SDRES01 MQW2BRK
  due to ACC(CONTROL or ALTER) access to: MQADMIN CSQ2.RESLEVEL

-------------------------------------------------------------------------------
....
```

MQCHECK REXX script

The REXX script MQCHECK that is shown in Example C-24 performs security checks and is especially useful in identifying WebSphere MQ channel initiator (CHIN) security vulnerabilities that are disabling proper authorization procedures.

Example C-24 REXX script to perform simple security checks

```
/*------------------------- rexx -------------------------------------*/
/* MQCHECK is for doing basic security checking on z/OS              */
/*                                                                    */
/* This program is doing some security checking based on best        */
/* practice as described in IBM Redbooks publication "Secure Messaging*/
/* Scenarios with WebSphere MQ". It is not the intention to write    */
/* a complete tool for checking your WebSphere MQ setup on z/OS.    */
/* This was intended to act as a start framework for inspiration     */
```

```
/* and for easing your daily security controls around WebSphere MQ.   */
/*                                                                    */
/* Invocation:                                                        */
/*   Syntax   %MQCHECK [ssid[-DDNAME]]                                */
/*           ssid is the name of the MQ subsystem either qmgr or QSG */
/*           DDNAME is the DDNAME that represents input from save-   */
/*           queue manager.                                           */
/*                                                                    */
/* If MQCHECK is started without parameters it will check the RACF   */
/* profiles for all defined WebSphere MQ systems on your LPAR         */
/*                                                                    */
/* If MQCHECK is started with ssid it will only check racf profiles  */
/* related to this queue manager.                                     */
/*                                                                    */
/* If MQCHECK is started with ssid-DDNAME it will investigate the    */
/* queue manager definition, channel definitions and print a report */
/* covering the state of your system.                                 */
/*                                                                    */
/*--------------------------------------------------------------------*/
parse upper arg ssid '-' ddname .
parse upper arg parms                       /* for heading only     */

Numeric digits 20               /* default of 9 digits is not enough */
call common                                 /* set some comon ptrs  */
ssid=wstrip(ssid)                           /* strip ssid for ''s   */
say ""
say center(" MQCHECK v1r0m0 ",78,"*")
say "Executed on:" GRSNAME "by" userid() "date:" date() time() ,
    "Opt:" parms
call LINES
call AUTH                                   /* add disclamer        */
QmgrInfo._Status = 0                        /* No queue manager inf.*/
ChlInfo._Status = 0                         /* No Channel info.     */
ChlInfo._Count  = 0                         /* No Channels          */
ChlAuth._Count  = 0                         /* No Channel Auth.     */
if ddname <> "" then do
  ddname=wstrip(ddname)                     /* clear off ''''       */
  call LOADQMGRINFO ddname                  /* Load qmgr info from  */
                                            /* prev. SAVEQMGR step  */
  End
/* The following lines can show status from loading queue manager    */
/* definitions. This is intended for debugging purpose.              */
/*
say  'ChlInfo._Count  ' ChlInfo._Count
say  'ChlAuth._Count  ' ChlAuth._Count
say  'QmgrInfo._Status' QmgrInfo._Status
*/

/* See if migration has been done, we parse on info about qmgr status*/
if QmgrInfo._Status = 1 then
   call MIGRATIONCHECK 1 QmgrInfo._ScyCase QmgrInfo._PSMode
else
   call MIGRATIONCHECK 0

if ssid = "" then do
```

```
  SSCVT    = C2d(Storage(D2x(JESCT+24),4))   /* point to SSCVT       */
  NotAuth  = 0                               /* Access ok.           */
  Do while SSCVT <> 0
    SSCTSNAM = Storage(D2x(SSCVT+8),4)       /* subsystem name       */
    SSCTSUSE = C2d(Storage(D2x(SSCVT+20),4)) /* SSCTSUSE pointer     */

    if SSCTSUSE > 0 then do
      SSCTSUSE_check = left(right(d2x(SSCTSUSE),8,'0'),1)
      if SSCTSUSE_check >= '0' & SSCTSUSE_check < '8' then Do
        if Storage(D2x(SSCTSUSE+4),4) = "ERLY" &  ,
          Storage(D2x(SSCTSUSE+84),8) = "CSQ3EPX " then do
          call CHCKRACF SSCTSNAM             /* Execute RACF on qmgr */
          end
        end
      end
  SSCVT    = C2d(Storage(D2x(SSCVT+4),4))    /* next sscvt or zero   */
  End                                        /* Do while SSCVT <> 0  */
End
else do
  call CHCKRACF ssid
  if QmgrInfo._Status = 1 then do
    call SSLREPORT ssid
    call CHLAUTHREPORT ssid
    End
  if ChlInfo._Status = 1 then do
    call CHLREPORT ssid
    End
End
exit 0

COMMON:             /* Control blocks needed by multiple routines   */
CVT      = C2d(Storage(10,4))              /* point to CVT         */
JESCT    = C2d(Storage(D2x(CVT + 296),4))  /* point to JESCT       */
CVTSNAME = Storage(D2x(CVT + 340),8)       /* point to system name */
GRSNAME  = Strip(CVTSNAME,'T')             /* del trailing blanks  */
return


/*-------------------------------------------------------------*/
/* Check Queue manager security.                               */
/*                                                             */
/* Currently the following profiles are checked:               */
/* qmgr.RESLEVEL for qmgrCHIN, this shal have access NONE      */
/*-------------------------------------------------------------*/

CHCKRACF: procedure expose NotAuth
parse upper arg ssid

Reslevel_error = 0
Reslevel_found = 0

say ""
say "Validating queue manager:" ssid

/* Start to see if subsystem security has been switched off      */
```

```
noss = CheckSwProfile(ssid "NO.SUBSYS.SECURITY" 0)
if noss = 1 then do
  call WARN
  do i = 1 to 5
    say "ALL SECURITY IS SWITCHED OFF. NO FURTHER CHECKS MADE" ,
        ssid
    End
  return
  End


if NotAuth = 1 then do               /* We have not access     */
  say "  Check not performed."
  Return                             /*   just bail out :o(    */
  End


/*-------------------------------------------------------------*/
/* Start asking RACF about which user is running our CHIN task */
/*-------------------------------------------------------------*/

class = 'STARTED'

x = OUTTRAP('StartCl.',100)

/* "RLIST STARTED qmgr|allCHIN STDATA" */

"RLIST " class   SSID||"CHIN STDATA"
x = OUTTRAP('OFF')                     /* turns trapping OFF */

S_Profile = ""
Username= ""

/*-------------------------------------------------------------*/
/* Analyze the result from searching STARTED class in          */
/* RACF for <QMGR>CHIN userid                                  */
/*-------------------------------------------------------------*/
if StartCl.0 < 5 then do         /* Nothing found or auth miss */
  say " " StartCl.1
  say " " StartCl.2
  say "  Check not performed."
  if word(StartCl.2,1) = "IRR52021I" then do
    say ""
    say "*** MQCHECK should be executed by a security",
        "administrator"
    NotAuth = 1                  /* Don't check more STCs      */
    End
  return                         /* bail out, don't check more */
  End
else Do
  Class_lid=0
  do i= 1 to StartCl.0
    parse var StartCl.i w1 w2 w3 w4 w5 w6
    if left(w1,1) = "-" then iterate
    if w1 = 'CLASS' then Do
```

```
      Class_lid=i+2           /* profile will be on +2 */
      End
    if w1 = 'USER=' then Do
      Username=w2          /* Username of started task */
      iterate
      End
    if i=Class_lid then Do
      Profile=w2 w3
      iterate
      End
    End
  End

/* Report result of analyze of STARTED Class           */
say "  "||SSID||"CHIN run under" username

/*-------------------------------------------------------------*/
/* Obtain what information of what groups chin user is         */
/* connected to.                                               */
/*-------------------------------------------------------------*/

x = OUTTRAP('UserInfo.',1000)

/* "LU userid" */

"LU "username
x = OUTTRAP('OFF')              /* turns trapping OFF */
if UserInfo.0 = 1 then
  say UserInfo.1
else Do
  Groups  = ""
  do i= 1 to UserInfo.0
    parse var UserInfo.i w1 w2 w3 w4 w5 w6 1 c1 '=' c2 .
    if c1 = "GROUP" then Do
       Groups = Groups||c2||" "
       End
    End
  End

rc=wspace(4 "User" username "is connected to" groups)

/*-------------------------------------------------------------*/
/* Obtain MQADMIN information about the RESLEVEL for the CHIN  */
/* user                                                        */
/*-------------------------------------------------------------*/

class = 'MQADMIN'
x = OUTTRAP('Reslvl.',100)

/* "RLIST MQADMIN  SSID.RESLEVEL ALL" */

"RLIST " class   ssid".RESLEVEL ALL"
x = OUTTRAP('OFF')                      /* turns trapping OFF */
if Reslvl.0 = 1 then
  say Reslvl.1
```

```
else Do
  Profile = ""
  Univ_acc= ""
  Class_lid=0
  Univ_lid=0
  Users_con=""
  Users_acc=""
  Users_cf=0
  do i= 1 to Reslvl.0
    parse var Reslvl.i w1 w2 w3 w4 w5 w6
    if left(w1,1) = "-" then iterate

    if w1 = 'CLASS' then Do
      Class_lid=i+2                   /* profile will be on +2 */
      End
    if w1 = 'LEVEL' then Do
      Univ_lid=i+2             /* Universal auth will be on +2 */
      End
    if w1 = 'ID' & w2 ='ACCESS' & w3 = 'ACCESS' then Do
      Users_cf=0                                     /* Done */
      End
    if (w1 = 'USER') & (w2 ='ACCESS') & (w3 = 'ACCESS') then Do
      Users_cf=i+1             /* Connected groups will be +2  */
      End

    /* Grap the results now                                    */
    if i=Class_lid then Do
      Profile=w2 w3
      iterate
      End
    if i=Univ_lid then Do
      Univ_acc=w3
      iterate
      End
    if (users_cf > 0) & (i > Users_cf) then Do
      if strip(Reslvl.i) <> "NO USERS IN ACCESS LIST" then do
        Users_con = Users_con||w1||" "
        Users_acc = Users_acc||w2||" "
        End
      iterate
      End
    End
  End


/*-------------------------------------------------------------*/
/* Now we will analyze the obtained information and produce a  */
/* rapport based on this information                           */
/*-------------------------------------------------------------*/
ReslevelSec_Con = 0                          /* we assume ok   */
ReslevelSec_txt = ""                         /* we assume ok   */
ReslevelSecBadUsr=""
BadAccessChin   = "CONTROL ALTER"    /* Accesses we don't like */

if users_con <> "" then
```

```
  say "  Users/groups connected to" strip(profile) ":" users_con

/* Now see if we have a match on a group with access           */
/* qmgr.RESLEVEL with access better than UPDATE                */
userpos= wordpos(username,users_con)
if userpos > 0 then do
  if wordpos(word(users_acc,userpos),BadAccessChin) > 0 then do
    ReslevelSec_Con = 1        /* We got a match, mark it      */
    ReslevelSec_txt = "User" username "is on access list:" ,
      profile "with ACC("||word(users_acc,userpos)||") on" ,
      class profile
    End
  End

/* Now see if the CHIN user is on an access list with high     */
/* access rights.                                              */

Do i=1 to words(Users_con)
  CheckGrp=word(Users_con,i)
  if wordpos(CheckGrp, Groups) > 0 then do /* Chin membership  */
    if wordpos(word(users_acc,i),BadAccessChin) > 0 then do
      ReslevelSec_Con = 2    /* We got a match, mark it        */
      ReslevelSec_txt = "User" username "is connected to:" ,
        CheckGrp "with ACC("||word(users_acc,i)||") on" ,
        class profile
      End
    End
  End

/* Now see check the users connected groups and match them     */
/* against the groups on the access list                       */

Do i=1 to words(Users_con)
  CheckGrp=word(Users_con,i)
  if wordpos(word(users_acc,i),BadAccessChin) > 0 then do
    ReslevelSecBadUsr = ReslevelSecBadUsr " " CheckGrp
    End
  End

/* Now check the Bad users, see if it's a group, and if so     */
/* Expand it.                                                  */

ReslevelSecBadUsrResult=""
Do j=1 to words(ReslevelSecBadUsr)

  group = word(ReslevelSecBadUsr,j)
  x = OUTTRAP('GroupInfo.',1000)

  /* "LG group" */

  "LG "group
  x = OUTTRAP('OFF')              /* turns trapping OFF        */

  if GroupInfo.0 = 1 then do      /* we didn't get any useful */
    if left(GroupInfo.1,31) = ,
```

```
            "NAME NOT FOUND IN RACF DATA SET" then do
      ReslevelSecBadUsrResult = ReslevelSecBadUsrResult " " group
      End
    else do
    say GroupInfo.1
      End
    End
  else Do
    do i= 1 to GroupInfo.0
      parse var GroupInfo.i w1 . 1 c1 6 c2 9 .
      if c1 = "      " & (c2 <> "   ") then Do
         ReslevelSecBadUsrResult = ReslevelSecBadUsrResult " " w1
         End
      End
    End
  End

/* Report status of queue manager security                    */
if univ_acc = "NONE" & ReslevelSec_Con = 0 then do
  rc=wspace(2 "Queue manager:" SSID class Profile ,
            "UACC("univ_acc") seems to be fine")
  say "  for CHIN security. Recommended setting."
  End
else
  rc=wspace(2 "Queue manager:" SSID class Profile ,
            "UACC("univ_acc")")
if univ_acc <> "NONE" then Do
  call WARN
  say "Channel security is turned off due to universal access to",
      class profile
  End
if ReslevelSec_Con>0 then do
  call WARN
  say "Channel security is turned off due to granted access:"
  rc=wspace(2 ReslevelSec_txt)
  End

if ReslevelSecBadUsrResult <> "" then do
  call WARN
  Say "The following users have security checks disabled:"
  rc=wspace(2 ReslevelSecBadUsrResult)
  say "  due to ACC(CONTROL or ALTER) access to:" class Profile
  End

if noss = 0 then do
  call CheckSwProfile( ssid "NO.CMD.CHECKS")
  call CheckSwProfile( ssid "NO.CMD.RESC.CHECKS")
  call CheckSwProfile( ssid "NO.CONTEXT.CHECKS")
  call CheckSwProfile( ssid "NO.ALTERNATE.USER.CHECKS")

  call CheckClassUacc "MQCMDS"  ssid "ALTER.QMGR"
  call CheckClassUacc "MQQUEUE" ssid "SYSTEM.COMMAND.QUEUE"
  call CheckClassUacc "MQQUEUE" ssid "SYSTEM.ADMIN.COMMAND.QUEUE"
  call CheckClassUacc "MXQUEUE" ssid "SYSTEM.COMMAND.QUEUE"
  call CheckClassUacc "MXQUEUE" ssid "SYSTEM.ADMIN.COMMAND.QUEUE"
```

```
  call CheckClassUacc "MXTOPIC" ssid "PUBLISH.SYSTEM.BASE.TOPIC"
  call CheckClassUacc "MXTOPIC" ssid "SUBSCRIBE.SYSTEM.BASE.TOPIC"
  End

return

/*-----------------------------------------------------------------*/
/* Generic check of class for queue manager to see if UACC has */
/* been set to other than 'NONE'                               */
/*-----------------------------------------------------------------*/
CheckClassUacc: procedure
parse arg class ssid cprofile

x = OUTTRAP('Reslvl.',100)

/* "RLIST MXTOPIC SSID.SUBSCRIBE.SYSTEM.BASE.TOPIC generic all" */

"RLIST " class   ssid"."cprofile" GEN ALL"
x = OUTTRAP('OFF')                        /* turns trapping OFF */
if Reslvl.0 < 5 then do
  if word(Reslvl.1,1) = "IKJ56702I" then do  /* Class not def.  */
    Say "RACF Class:" class "is not defined."
    Return
    End
  say Reslvl.1
  Return
  End
else Do
  Profile = ""
  Univ_acc= ""
  Class_lid=0
  Univ_lid=0
  Users_con=""
  Users_acc=""
  Users_cf=0
  do i= 1 to Reslvl.0
    parse var Reslvl.i w1 w2 w3 w4 w5 w6
    if left(w1,1) = "-" then iterate

    if w1 = 'CLASS' then Do
      Class_lid=i+2                    /* profile will be on +2 */
      End
    if w1 = 'LEVEL' then Do
      Univ_lid=i+2              /* Universal auth will be on +2 */
      End

    /* Grap the results now                                     */
    if i=Class_lid then Do
      Profile=w2 w3
      iterate
      End
    if i=Univ_lid then Do
      Univ_acc=w3
      iterate
      End
```

```
    End
  End

if Univ_acc <> "NONE" then do
  call WARN
  rc=wspace(0 "There might be security issues with class" ,
            class " for " profile)
  rc=wspace(2 "located from:" cprofile "UACC("||Univ_acc||")")
  End

return

/*-----------------------------------------------------------------*/
/* Generic check of switch profiles in MQADMIN class.              */
/*-----------------------------------------------------------------*/
CheckSwProfile: procedure
parse arg ssid cprofile trc
rc = 1                                  /* We have no problems */

x = OUTTRAP('Reslvl.',100)

/*"RLIST MXTOPIC SSID.SUBSCRIBE.SYSTEM.BASE.TOPIC generic all" */

class="MQADMIN"
"RLIST " class   ssid"."cprofile" ALL"
x = OUTTRAP('OFF')                       /* turns trapping OFF */
if Reslvl.0 = 1 then
  say Reslvl.1
else Do
  Profile = ""
  Univ_acc= ""
  Class_lid=0
  Univ_lid=0
  Users_con=""
  Users_acc=""
  Users_cf=0
  do i= 1 to Reslvl.0
    parse var Reslvl.i w1 w2 w3 w4 w5 w6
    if left(w1,1) = "-" then iterate

    if trc=1 then say Reslvl.i

    if w1 = 'CLASS' then Do
      Class_lid=i+2                   /* profile will be on +2 */
      End
    if w1 = 'LEVEL' then Do
      Univ_lid=i+2             /* Universal auth will be on +2 */
      End

    /* Grap the results now                                     */
    if i=Class_lid then Do
      Profile=w2 w3
      iterate
      End
    if i=Univ_lid then Do
```

```
      Univ_acc=w3
      iterate
      End
    End
  End

if (Univ_acc <> "NONE") & (left(profile,2) <> "**") then do
  call WARN
  say "There might be security issues with class" class " for ",
      profile
  say "Located from:" cprofile
  rc=1                                    /* We have a problem*/
  End

return rc

/*-------------------------------------------------------------*/
/* Perform a migration to mixed case profile to ensure         */
/* security is enabled on PUBSUB because SCK(UPPER) will not   */
/* perform security checks against MXTOPIC and therefore could */
/* lead to security vulnerabilities.                           */
/* This check is not water proof, but based on guessing that   */
/* RACF classes MXADMIN and MXQUEUE should contain the same or */
/* more profiles than MQADMIN and MQQUEUE. If not MQCHECK will */
/* raise a warning if MXTOPIC contains profiles.               */
/*                                                             */
/* NoInfo: 0 : No queue manager available, run system check    */
/*         1 : Queue manager data available, if SCYCASE is      */
/*             we're deailing with a pre version 7 queue       */
/*             manager.                                        */
/*                                                             */
/*-------------------------------------------------------------*/
MIGRATIONCHECK: procedure expose GRSNAME QmgrInfo.
parse arg NoInfo ScyCase PSMode


/* Count profiles in CLASS MXADMIN                             */
MXADMINClass = CountClassProfiles("MXADMIN")

/* Count profiles in CLASS MQADMIN                             */
MQADMINClass = CountClassProfiles("MQADMIN")

/* Count profiles in CLASS MXTOPIC                             */
MXTOPICClass = CountClassProfiles("MXTOPIC")

/* Count profiles in CLASS MXQUEUE                             */
MXQUEUEClass = CountClassProfiles("MXQUEUE")

/* Count profiles in CLASS MQQUEUE                             */
MQQUEUEClass = CountClassProfiles("MQQUEUE")

/* Count profiles in CLASS MQQUEUE                             */
MQCMDSClass = CountClassProfiles("MQCMDS")

/* Count profiles in CLASS MXNLIST                             */
```

```
MXNLISTClass = CountClassProfiles("MXNLIST")

/* Count profiles in CLASS MXPROC                                */
MXPROCClass = CountClassProfiles("MXPROC")

ClassesNotDefined = ""
if MXQUEUEClass = -2 then do
  ClassesNotDefined = ClassesNotDefined "MXQUEUE "
  MXQUEUEClass = -1
  End
if MXADMINClass = -2 then do
  ClassesNotDefined = ClassesNotDefined "MXADMIN "
  MXADMINClass = -1
  End
if MXTOPICClass = -2 then do
  ClassesNotDefined = ClassesNotDefined "MXTOPIC "
  MXTOPICClass = -1
  End
if MXPROCClass = -2 then do
  ClassesNotDefined = ClassesNotDefined "MXPROC "
  MXPROCClass = -1
  End
if MXNLISTClass = -2 then do
  ClassesNotDefined = ClassesNotDefined "MXNLIST "
  MXNLISTClass = -1
  End

say "Statistics from class query performed on:" GRSNAME
say ""
say "                MXADMIN  MQADMIN  MXQUEUE  MQQUEUE",
                  " MXTOPIC   MQCMDS"
say "             "  right(MXADMINClass,8)  ,
                     right(MQADMINClass,8)  ,
                     right(MXQUEUEClass,8)  ,
                     right(MQQUEUEClass,8)  ,
                     right(MXTOPICClass,8)  ,
                     right(MQCMDSClass,8)

if ClassesNotDefined <> "" then do
  say "The following RACF classes are not defined:"
  say " " space(ClassesNotDefined)
  say ""
  End

if MQQUEUEClass = -1 then do /* It seems no access             */
  say ""
  say "MQCHECK was unable to obtain class information from",
      "RACF, ask security administration to run job."
  return                                  /* Bail out          */
  End

if NoInfo = 0 then do          /* No SAVEQMGR info available   */
  if MQQUEUEClass > 0  & ,     /* Assure we have access        */
    (MXTOPICClass = -1 | ,
     MXADMINClass = -1 | ,
```

```
     MXQUEUEClass = -1 ) then do
    say ""
    call NOTE
    say "Mixed case security classes seems not to be defined",
        "in RACF please make sure"
    say "that PSMODE(DISABLED) is specified on QMGR object in",
        "your queue managers."
    say "If PSMODE is either COMPAT or ENABLED you have might",
        "have a security"
    say "vulnerability."
    say "See MQ information center for further information."
    call LINE
    return                                  /* Bail out        */
    End

/* Now we're guessing about if migration to MX... classes has  */
/* The guess is based on comparison of number of classes      */
/* defined in MX and MQ classes....                            */
  if MQQUEUEClass > 0  & ,     /* Assure we have access        */
    (MXQUEUEClass > 0  & ,
     MXADMINClass > 0  & ,
    (MXQUEUEClass < MQQUEUEClass | ,
     MXADMINClass < MQADMINClass ) ) then do
    say ""
    call NOTE
    say "It seems like you have NOT switched ""Mixed case",
        "security classes"" on, but have"
    say "defined MX-classes without copying the MQ-classes over."
    say "Please assure that you have proper PUBSUB security in",
        "place."
    say "If SCYCASE(UPPER) is in use, it means that there are",
        "no security on your PUBSUB|"
    say "MQ will NOT check against MXTOPIC when SCYCASE(UPPER)",
        "is in use."
    call LINE
    return                                      /* Bail out */
    End
  return
  End                          /* end of system check          */


if NoInfo = 1 then do          /* No SAVEQMGR info available   */

  /* See if PUBSUB is specified, if not don't care pre v.7 qmgr*/
  if QmgrInfo._PSMode = '' then do
    say "Not a version 7 queue manager."
    say "  No MX class checks are performed."
    return                                      /* Bail out */
    End

  /* See if PUBSUB is disabled, if so don't care               */
  if QmgrInfo._PSMode = 'DISABLED' then do
    say "PUBSUB is disabled on this queue manager"
    say "  No MX class checks are performed."
    return                                      /* Bail out */
```

```
    End

  /* PUBSUB should be enabled here                              */
  if QmgrInfo._PSMode = 'ENABLED' | ,
     QmgrInfo._PSMode = 'COMPAT' then do
    say "PUBSUB is enabled, PUBSUB Security need a check."
    if QmgrInfo._ScyCase = "UPPER" then do /* We have a problem*/
      call WARN
      do i=1 to 5
        Say "MXTOPIC profiles are is NOT used.",
            "PUBSUB is not protected."
        End
      call LINES
      Return                                        /* bail out */
      End
    if QmgrInfo._ScyCase = "MIXED" then do /* Seems to be OK   */
      Return                                        /* bail out */
      End
    End
  End                           /* end of system check          */
return

/*-------------------------------------------------------------*/
/* Check global SSL/TLS security settings on queue manager     */
/*-------------------------------------------------------------*/
SSLREPORT: procedure expose QmgrInfo.
parse arg ssid

say ""
if QmgrInfo._SSLKeyR = "' '" then do  /* We are not using SSL  */
  say "SSL/TLS are not used. Authentication should",
      "be considered."
  say "See Infocenter for additional information"
  End
Else do                          /* SSL/TLS is enabled, CRL?  */
  if QmgrInfo._SSLCrlNl = "' '" then do  /* No CRL, report it  */
    say "No Certification Revocation List(CRL) controls are in",
        "place, to guarantee"
    say "certificates are still trustworthy."
    End
  Else do
    say "SSL/TLS keyring settings seems to be fine."
    End
  End
return


/*-------------------------------------------------------------*/
/* Check global CHLAUTH security is enabled on queue manager   */
/*-------------------------------------------------------------*/
CHLAUTHREPORT: procedure expose QmgrInfo. ChlAuth.
parse arg ssid

ChlAuth = wstrip(QmgrInfo._ChlAuth)
if ChlAuth = "" then do               /* Pre version 7.1 qmgr  */
```

```
  Return
  End

if QmgrInfo._ChlAuth = "ENABLED" then do /* Enabled is fine    */
  say "Channel authentication is enabled on queue manager" ssid
  say "  rules should be validated."
  End
Else do                           /* SSL/TLS is enabled, CRL?  */
  say "Channel authentication is not enabled. CHLAUTH should",
      "be enabled"
  say "according to best practice."
  End

/* here you can add processing for CHLAUTH rules and validating*/
/* with the channels                                           */
/* or you can place the validating in the channel validation   */

/*  say "Processing CHLAUTH records"                           */
/*do i = 1 to ChlAuth._Count                                   */
/*  say ChlAuth._mcauser.i space(ChlAuth._userlist.i)          */
/*  End                                                        */



return

/*-------------------------------------------------------------*/
/* Analyze the channel settings according to standard hardening*/
/*-------------------------------------------------------------*/
CHLREPORT: procedure expose ChlInfo.

call LINE
say ""
say "Channel security report:"
Problems       = 0
WantedChlTypes = "SVRCONN RCVR RQSTR CLUSRCVR SVR"
UnOpenChls     = "SYSTEM.DEF.SVRCONN  SYSTEM.DEF.CLUSRCVR" ,
                 "SYSTEM.DEF.RECEIVER SYSTEM.DEF.REQUESTER" ,
                 "SYSTEM.AUTO.RECEIVER SYSTEM.AUTO.SVRCONN" ,
                 "SYSTEM.DEF.SERVER"
do ix = 1 to ChlInfo._Count   /* Process collected channels    */
/* We check inbound channels only                            */
  if wordpos(ChlInfo._chltype.ix, WantedChlTypes) > 0 then do
/*
 say ChlInfo._chlname.ix ChlInfo._chltype.ix
 say ChlInfo._scyexit.ix ChlInfo._sslcauth.ix ChlInfo._sslciph.ix
 say ChlInfo._sslpeer.ix ChlInfo._mcauser.ix  ChlInfo._maxmsgl.ix
*/
    if (wordpos(wstrip(ChlInfo._chlname.ix),UnOpenChls)) > 0 ,
       then do
      if ChlInfo._mcauser.ix =  "' '" | ,
         ChlInfo._sslciph.ix =  "' '" | ,
         ChlInfo._maxmsgl.ix >  1 then do
        Problems = Problems + 1
        say ""
```

```
        say "Channel:" wstrip(ChlInfo._chlname.ix) "should",
            "be disabled to prevent unauth. usage."
        say "  MCAUSER() and/or SCYEXIT() is blank or",
            "MAXMSGL > 1"
        iterate
        End
      End
    if ChlInfo._sslciph.ix =  "' '" & ,
       ChlInfo._scyexit.ix =  "' '" & ,
       ChlInfo._mcauser.ix =  "' '" then do
      Problems = Problems + 1
      say ""
      say "Channel:" wstrip(ChlInfo._chlname.ix) "is not",
          "protected according to best practice,"
        say "  MCAUSER(), SCYEXIT() and SSLCIPH() are all blank."
      iterate
      End
    if ChlInfo._sslciph.ix =  "' '" & ,
       ChlInfo._scyexit.ix =  "' '" & ,
       ChlInfo._mcauser.ix <> "' '" then do
      Problems = Problems + 1
      say ""
      say "Channel:" wstrip(ChlInfo._chlname.ix),
          "might not be sufficient secured,"
      say "  check that MCAUSER('"||,
          wstrip(ChlInfo._mcauser.ix)||,
          "') have no administrative rights."
      iterate
      End
    if ChlInfo._sslciph.ix =  "' '" & ,
       ChlInfo._scyexit.ix <> "' '" then do
      Problems = Problems + 1
      say ""
      say "Channel:" wstrip(ChlInfo._chlname.ix) ,
          "assure that security exit:" ,
          wstrip(ChlInfo._scyexit.ix)
      say "  perform real validation, to prevent",
          "unauthorized usage."
      iterate
      End
    End
  End
if Problems = 0 then do
  say "  No channel problems was found."
  End

return

/*-----------------------------------------------------------*/
/* strip input for apostrophes and leading/trailing spaces       */
/*-----------------------------------------------------------*/
wstrip: procedure
parse arg input
return strip(translate(input,' ',"'"))
```

```
/*-------------------------------------------------------------*/
/* Count number of class entries in class                      */
/* Returns -1 incase of an error                               */
/*-------------------------------------------------------------*/
CountClassProfiles: procedure
parse arg class .

Count = -1

x = OUTTRAP('Reslvl.',100000)

/* "RLIST MXTOPIC *"                                           */

"RLIST " class "*"
x = OUTTRAP('OFF')             /* turns trapping OFF           */
if Reslvl.0 = 1 then do
  if word(Reslvl.1,1) = "ICH13004I" then do
    Count = -2                 /* Class is not in place        */
    Return Count               /* Bail out, problem found      */
    End
  say Reslvl.1
  End
else Do
  if word(Reslvl.1,1) = "INVALID" then do
    Return Count               /* Bail out, class not defined  */
    End
  if Reslvl.0 < 5 then do
    Return Count               /* Bail out, problem found      */
    End
  Count = 0
  do i= 1 to Reslvl.0
    parse var Reslvl.i w1 w2 .
    if left(w1,1) = "-" then iterate
    if w1 = "CLASS" & w2 = "NAME" then           /* Just count */
      Count = Count + 1                          /*   classes  */
    End
  End
return Count

/*-------------------------------------------------------------*/
/* Load information retrieve from the previous SAVEQMGR        */
/*                                                             */
/* Parameters: DDNAME to read from                             */
/*                                                             */
/* It returns it's result in QmgrInfo. variables.              */
/*                                                             */
/*-------------------------------------------------------------*/
LOADQMGRINFO: procedure expose QmgrInfo. ChlInfo. ChlAuth.
parse arg ddname .

x = OUTTRAP('LoadQI.',1)

wline      = ""
do forever
  /* Load file into stem variable line.1 one record at a time  */
```

```
  'EXECIO 1 DISKR' ddname '(STEM line.'
  if RC = 0 then do
    parse var line.1 1 pos1 7 1 chr1 2
    if chr1 = "*"      then iterate
    if pos1 = "DEFINE" | ,      /* New object detected, save   */
       pos1 = "ALTER " | ,      /*   what we have              */
       pos1 = "SET   " then do
      if wline <> '' then do    /* Is buffer in use ?          */
        call PROCDESSLINE wline
        wline = ""              /* Clear it                    */
        End
      End
    wline = wline " " strip(line.1)  /* Concat new stuff       */
    End
  else do
    if wline <> '' then do      /* Is buffer in use ?          */
      call PROCDESSLINE wline
      End
    leave
    End
  End
'EXECIO 0 DISKR' ddname '(FINIS'
return

/*-----------------------------------------------------------*/
/* Decode content of wline and update QmgrInfo. or ChlInfo.    */
/*-----------------------------------------------------------*/
PROCDESSLINE: procedure expose QmgrInfo. ChlInfo. ChlAuth.
parse arg wline

parse var wline w1 w2 . 1 . . w3 '('
/* handle the record types                                     */
w3 = strip(w3,'B')
select
  when w1 = "ALTER" & w2 = "QMGR" then do
    /* We will just take some queue manager properties         */
    parse var wline 1 'QSGNAME('  QmgrInfo._CSQName  ')' ,
                    1 'PSMODE('   QmgrInfo._PSMode   ')' ,
                    1 'DESCR('    QmgrInfo._Descr    ')' ,
                    1 'CHLAUTH('  QmgrInfo._ChlAuth  ')' ,
                    1 'SCYCASE('  QmgrInfo._ScyCase  ')' ,
                    1 'SSLKEYR('  QmgrInfo._SSLKeyR  ')' ,
                    1 'SSLCRLNL(' QmgrInfo._SSLCrlNl ')' ,
                    1 'DEADQ('    QmgrInfo._DeadQ    ')'

    if QmgrInfo._DeadQ <> "" then
       QmgrInfo._Status = 1      /* We found something we know */
    End
  when w1 = "DEFINE" then do
    /* We will just take some channel properties               */
    parse var wline 1 'CHANNEL('  chlname  ')' ,
                    1 'CHLTYPE('  chltype  ')' ,
                    1 'SCYEXIT('  scyexit  ')' ,
                    1 'SSLCAUTH(' sslcauth ')' ,
                    1 'SSLCIPH('  sslciph  ')' ,
```

```
                    1 'SSLPEER('  sslpeer  ')' ,
                    1 'MCAUSER('  mcauser  ')' ,
                    1 'MAXMSGL('  maxmsgl  ')'
    ix = ChlInfo._Count + 1
/*  say chlname chltype mcauser sslciph ix */
    ChlInfo._Status     = 1        /* Set flag                 */
    ChlInfo._Count      = ix       /* point to next element    */
    ChlInfo._wline.ix   = wline    /* save total line DELETE IT*/
    ChlInfo._chlname.ix = chlname  /* save parsed chl attrs    */
    ChlInfo._chltype.ix = chltype
    ChlInfo._scyexit.ix = scyexit
    ChlInfo._sslcauth.ix= sslcauth
    ChlInfo._sslciph.ix = sslciph
    ChlInfo._sslpeer.ix = sslpeer
    ChlInfo._mcauser.ix = mcauser
    ChlInfo._maxmsgl.ix = maxmsgl
    End
  when w1 = "SET" & w3 = "CHLAUTH" then do
    /* We will just take some channel properties            */
    parse var wline 1 'CHLAUTH('  chlname  ')' ,
                    1 'TYPE('     chlatype ')' ,
                    1 'ADDRESS('  chladdr  ')' ,
                    1 'USERLIST(' chlusrl  ')' ,
                    1 'USERSRC('  chlusrc  ')' ,
                    1 'MCAUSER('  chlmcau  ')'
    ix = ChlAuth._Count + 1
    ChlAuth._Count      = ix       /* point to next element    */
    ChlAuth._chlauth.ix = chlname
    ChlAuth._type.ix    = chlatype
    ChlAuth._addrress.ix= chladdr
    ChlAuth._userlist.ix= chlusrl
    ChlAuth._usersrc.ix = chlusrc
    ChlAuth._mcauser.ix = chlmcau
    End
  otherwise do
    say "Unknown object received w1" w1 "w2"  w2 "w3" w3
    nop
    End
  End
return
/*-------------------------------------------------------------*/
LINE:
   say left("",78,"-")
return

/*-------------------------------------------------------------*/
LINES:
   say left("",78,"*")
return

/*-------------------------------------------------------------*/
NOTE:
   say ""
   call LINE
   say "NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE",
```

```
       "NOTE NOTE NOTE NOTE NOTE"
   say "NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE NOTE",
       "NOTE NOTE NOTE NOTE NOTE"
return


/*-------------------------------------------------------------*/
WARN:
   say ""
   call LINE
   say "WARNING WARNING WARNING WARNING WARNING WARNING WARNING",
       "WARNING WARNING WARNING"
return

/*-------------------------------------------------------------*/
/* Format the presented string so it fits between boundaries    */
/* and intend it.                                               */
/*-------------------------------------------------------------*/
wspace: procedure
parse arg intend text
wstr=strip(space(text))
if (length(wstr) + intend) < 78 then do /* Special formatting? */
  if intend > 0 then
    say left("",intend," ")||wstr     /*      Nope, just print */
  else
    say wstr
  End
Else do                             /* Yes, build lines...   */
  maxlen=79-intend
  wstr=""
  do while text <> ""
    parse var text nword text
    if length(strip(wstr))+length(nword)+1 > maxlen then do
      if intend > 0 then
        say left("",intend," ")||strip(wstr)    /* print line    */
      else
        say strip(wstr)                         /* print line    */
      wstr=""
      End
    wstr=wstr||nword||" "
    End
  if strip(wstr) <> "" then
    say left("",intend," ")||strip(wstr)    /* print line    */
  End
return 0


/*-------------------------------------------------------------*/
AUTH:
   say ""
   say "This program is based AS-IS and there are no guarantees",
       "that it finds all security"
   say "holes, breaches. This solely is intended to display issues",
       "about CHIN task"
   say "issues and on moving from uppercase classes (MQ...)",
```

```
        "mixed case classes (MX...)"
   say ""
   say "A user of this program should understand that IBM cannot",
        "provide technical"
   say "technical support for the program and will not be",
        "responsible for any"
   say "consequences of use of the program".
   say ""
   say "Regards"
   say "WebSphere MQ security Redbooks team 2012"
   say ""
   call LINE
return
/*-----------------------------------------------------------------*/
/* End of MQCHECK                                                  */
/*-----------------------------------------------------------------*/
```

D

Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the Web material

The Web material that is associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Point your web browser at:

ftp://www.redbooks.ibm.com/redbooks/SG248069

Alternatively, you can go to the IBM Redbooks website at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG248069.

Using the Web material

The additional Web material that accompanies this book includes the following files:

Folder name	*Description*
Chapter 9 scripts	Scripts used in Chapter 9, “Scenario: WebSphere MQ administration” on page 97
Appendix A scripts	Scripts used in Appendix A, “Working with the itsoME message exit” on page 263
Appendix B scripts	Scripts used in Appendix B, “Additional tooling for WebSphere MQ Internet pass-thru” on page 291
Appendix C scripts	Scripts used in Appendix C, “Certificate administration techniques and special WebSphere MQ security checks” on page 307

Downloading and extracting the Web material

Create a sub-directory (folder) on your workstation, and extract the contents of the Web material `.zip` file into this folder.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publication provides additional information about the topic in this document. Note that publications might be available in softcopy only.

- *WebSphere MQ V7.0 Features and Enhancements*, SG24-7583

You can search for, view, download or order these documents and other Redbooks, Redpapers, Web Docs, draft and additional materials, at the following website:

ibm.com/redbooks

Online resources

These websites are also relevant as further information sources:

- WebSphere MQ Security Bulletin

  http://ibm.co/WMQSecurityBulletin
- WebSphere MQ Recommended Fixes page

  http://ibm.co/WMQRecommendedFixes
- WebSphere MQ V7R5 Information Center

  http://pic.dhe.ibm.com/infocenter/wmqv7/v7r5/index.jsp
- WebSphere MQ V7R1 Information Center

  http://pic.dhe.ibm.com/infocenter/wmqv7/v7r1/index.jsp
- WebSphere MQ AMS V7.0.1 Information Center

  http://publib.boulder.ibm.com/infocenter/mqams/v7r0m1/index.jsp
- Fixes by version for WebSphere MQ

  http://www.ibm.com/support/docview.wss?uid=swg21254675
- Technote: MQMAXERRORLOGSIZE does not appear to increase the size of the AMQERR logs

  http://www.ibm.com/support/docview.wss?uid=swg21179310
- A Hursley view of WebSphere MQ

  http://hursleyonwmq.wordpress.com/
- MS0T: IBM WebSphere MQ Explorer

  http://www-01.ibm.com/support/docview.wss?uid=swg24021041

- MS81: WebSphere MQ Internet pass-thru

  http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24006386&loc=en_US&cs=utf-8&lang=en
- MQC75: WebSphere MQ V7.5 Clients

  http://www-304.ibm.com/support/docview.wss?uid=swg24032744
- WebSphere MQ - SupportPacs by Product

  http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg27007197#1
- Business Integration - WebSphere MQ SupportPacs

  http://www-01.ibm.com/support/docview.wss?rs=977&uid=swg27007205
- MS0P: WebSphere MQ Explorer - Configuration and Display Extension Plug-ins

  http://www-01.ibm.com/support/docview.wss?rs=171&uid=swg24011617&loc=en_US&cs=utf-8&lang=en
- MO71: WebSphere MQ for Windows - GUI Administrator

  http://www-01.ibm.com/support/docview.wss?uid=swg24000142
- MQ Support site

  http://www.ibm.com/software/integration/wmq/support/
- z/OS V1R12.0 Cryptographic Services System Secure Sockets Layer Programming z/OS V1R12.0

  http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=%2Fcom.ibm.zos.r12.gska100%2Fgska1a80.htm
- The z/OS V1R12.0 Security Server RACF Command Language Reference

  http://publib.boulder.ibm.com/infocenter/zos/v1r12/index.jsp?topic=%2Fcom.ibm.zos.r12.icha400%2Fichza4b0.htm
- IBM Support Assistant (ISA):

  http://www.ibm.com/software/support/isa/
- libcURL site

  http://curl.askapache.com/
- Microsoft TechNet

  http://technet.microsoft.com/en-us/
- Internet Engineering Task Force

  http://www.ietf.org
- OpenSSL ocsp

  http://www.openssl.org/docs/apps/ocsp.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Secure Messaging Scenarios with WebSphere MQ

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

Secure Messaging Scenarios with WebSphere MQ

Using strong authentication

Using granular authorization

Implementing revocation checking

The differences between well-designed security and poorly designed security are not always readily apparent. Poorly designed systems give the appearance of being secure but can over-authorize users or allow access to non-users in subtle ways. The problem is that poorly designed security gives a false sense of confidence. In some ways it is better to knowingly have no security than to have inadequate security believing it to be stronger than it actually is. But how do you tell the difference? Although it is not rocket science, designing and implementing strong security requires strong foundational skills, some examples to build on and the capacity to devise new solutions in response to novel challenges. This IBM Redbooks publication addresses itself to the first two of these requirements. This book is intended primarily for security specialists and WebSphere MQ administrators that are responsible for securing WebSphere MQ networks but other stakeholders should find the information useful as well.

Chapters 1 through 6 provide a foundational background for WebSphere MQ security. These chapters take a holistic approach positioning WebSphere MQ in context of a larger system of security controls including those of adjacent platforms technologies as well as human processes. This approach seeks to eliminate the simplistic model of security as an island, replacing it instead with the model of security as an interconnected and living system. The intended audience for these chapters includes all stakeholders in the messaging system from architects and designers to developers and operations.

Chapters 7 and 8 provide technical background to assist preparing and in configuring the scenarios and chapters 9 through 14 are the scenarios themselves. These chapters provide fully realized example configurations. One of the requirements for any scenario to be included was that it must first be successfully implemented in the team's lab environment. In addition, the advice provided is the cumulative result of years of participation in the online community by the authors and reflect real-world practices adapted for the latest security features in WebSphere MQ V7.1 and WebSphere MQ V7.5. Although these chapters are written with WebSphere MQ administrators in mind, developers, project leaders, operations staff and architects are all stakeholders who will find the configurations and topologies described here useful.

The third requirement mentioned in the opening paragraph was the capacity to devise new solutions in response to novel challenges. The only constant in the security field is that the technology is always changing. Although this book provides some configurations in a checklist format, these should be considered a snapshot at a point in time. It will be up to you as the security designer and implementor to stay current with security news for the products you work with and integrate fixes, patches or new solutions as the state of the art evolves.

SG24-8069-00 ISBN 0738437409

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks