

This task aims to establish security from a messaging application that transmits data to an IBM MQ application. How security will be implemented will be through encrypting whatever data is sent to the MQ using SSL/TLS.

First, when analysing our system network-wise, we see that data is sent bare (plain text). This makes our system susceptible to Man-in-the-middle-attacks or eavesdropping. For most legal jurisdictions data sent to be compliant to certain data protection policies such as PCI DSSI & GDPR.

So in the previous section after we managed to get the application to send data to our MQ. So when we execute the application this is the data we get in plain text.

```
Sent message:

JMSMessage class: jms_text
JMSType: null
JMSDeliveryMode: 2
JMSDeliveryDelay: 0
JMSDeliveryTime: 1716988239704
JMSExpiration: 0
JMSPriority: 4
JMSMessageID: ID:414d5120514d31202020202020202020d3f3566602960240
JMSTimestamp: 1716988239704
JMSCorrelationID: null
JMSDestination: queue:///DEV.QUEUE.1
JMSReplyTo: null
JMSRedelivered: false
  JMSXAppID: JmsPutGet (JMS)
  JMSXDeliveryCount: 0
  JMSXUserID: admin
  JMS_IBM_PutApplType: 28
  JMS_IBM_PutDate: 20240529
  JMS_IBM_PutTime: 13104097
Transaction:672

Received message:
Swift: FMBZBWGA2
SUCCESS
```

To test we opened WireShark and traced the traffic from our server.

Packet list	Narrow & Wide	Case sensitive	Regular Expression	admin	Find	Cancel
No. Time Source Destination Protocol Length Info						
10267 236.463778	192.168.1.111	34.180.209.148	HQ	394	HQOPEN	C.R=3.0 Type=MQOT-Q Obj=SYSTEM.PROTECTION.POLICY.QUEUE
10269 236.580955	192.168.1.111	34.180.209.148	TCP	74	[TCP Dup ACK 1025782] 54053 → 9443 [ACK] Seq=6251 Ack=348086 Win=131072 Len=0 SLE=346634 SRE=348086 SLE=343810 SRE=345222	
10273 236.655958	192.168.1.111	34.180.209.148	TCP	74	54053 → 9443 [ACK] Seq=6251 Ack=342398 Win=131072 Len=0 SLE=346634 SRE=348086 SLE=343810 SRE=345222	
10276 236.859553	192.168.1.111	34.180.209.148	HQ	178	REQUEST_70505	C.R=3.3 Hdl=0x0002 RqstFlags=0000000e GIDbgtGdx=0 MaxLen=1
10280 236.936213	192.168.1.111	34.180.209.148	TCP	66	54053 → 9443 [ACK] Seq=6251 Ack=345222 Win=131072 Len=0 SLE=346634 SRE=348086	
10282 237.086471	192.168.1.111	34.180.209.148	TCP	54	54053 → 9443 [ACK] Seq=6251 Ack=348086 Win=131072 Len=0	
10285 237.246935	192.168.1.111	34.180.209.148	TCP	54	54068 → 1414 [ACK] Seq=3901 Ack=3773 Win=32156 Len=0	
10286 237.247456	192.168.1.111	34.180.209.148	HQ	200	REQUEST_70505	C.R=3.3 Hdl=0x0002 RqstFlags=0000000e GIDbgtGdx=0 MaxLen=4096
10288 237.316941	192.168.1.111	34.180.209.148	TLSv1.2	96	Application Data	
10291 237.634857	192.168.1.111	34.180.209.148	TCP	54	54068 → 1414 [ACK] Seq=4049 Ack=3877 Win=32052 Len=0	
10292 237.639647	192.168.1.111	34.180.209.148	HQ	110	HQCLOSE	C.R=3.0 Hdl=0x0002
10293 237.659554	192.168.1.111	34.180.209.148	TCP	66	[TCP Dup ACK 1028821] 54053 → 9443 [ACK] Seq=6293 Ack=340455 Win=131072 Len=0 SLE=350870 SRE=352202	
10298 238.027351	192.168.1.111	34.180.209.148	HQ	926	HQPUT1	C.R=3.0 Type=MQOT-Q Obj=DEV.QUEUE.1 (Data 16 bytes)
10301 238.073620	192.168.1.111	34.180.209.148	TCP	74	[TCP Dup ACK 1028821] 54053 → 9443 [ACK] Seq=6293 Ack=349450 Win=131072 Len=0 SLE=353694 SRE=355106 SLE=350870 SRE=352202	
10302 238.116583	192.168.1.111	34.180.209.148	TCP	66	54053 → 9443 [ACK] Seq=6293 Ack=352282 Win=131072 Len=0 SLE=353694 SRE=355106	
10315 238.455538	192.168.1.111	34.180.209.148	TCP	54	54068 → 1414 [ACK] Seq=6377 Ack=4629 Win=32768 Len=0	
10325 238.424633	192.168.1.111	34.180.209.148	TCP	96	[TCP Dup ACK 1030281] 54053 → 9443 [ACK] Seq=6293 Ack=352282 Win=131072 Len=0 SLE=353694 SRE=355106	
10511 238.886881	192.168.1.111	34.180.209.148	TLSv1.2	96	Application Data	
10512 238.887028	192.168.1.111	34.180.209.148	TLSv1.2	96	Application Data	
10529 239.050470	192.168.1.111	34.180.209.148	TCP	66	54053 → 9443 [ACK] Seq=6377 Ack=4629 Win=32768 Len=0	
10533 239.265280	192.168.1.111	34.180.209.148	TCP	54	54053 → 9443 [ACK] Seq=6377 Ack=357930 Win=131072 Len=0	
10536 239.656464	192.168.1.111	34.180.209.148	TCP	66	[TCP Dup ACK 1053341] 54053 → 9443 [ACK] Seq=6377 Ack=357930 Win=131072 Len=0 SLE=360754 SRE=362166	
10539 240.275440	192.168.1.111	34.180.209.148	TCP	66	54053 → 9443 [ACK] Seq=6377 Ack=359342 Win=131072 Len=0 SLE=360754 SRE=362166	
10543 240.654087	192.168.1.111	34.180.209.148	TCP	54	54053 → 9443 [ACK] Seq=6377 Ack=363578 Win=131072 Len=0	
10550 241.620938	192.168.1.111	34.180.209.148	TCP	66	[TCP Dup ACK 1054381] 54053 → 9443 [ACK] Seq=6377 Ack=363578 Win=131072 Len=0 SLE=366487 SRE=367814	
Context...	0x00000000					
IndntCnt...	0					
UnidntCnt...	0					
IndntCnt...	0					
ResQName...	0					
ResQMgr...	0					
NumRecs...	0					
PMR Flag...	0x00000000					
OffsetPMR...	0					
OffsetRMR...	0					
AdriStrMR...	0x00000000					
AdriStrR...	0x00000000					
MQPUT/MQGET						
Rules and Formatting Header						
Structure:	BM					
version:	2					
Length:	156					
Encoding:	111-273 (FLT_IEEE_NORMAL/DEC_NORMAL/INT_NORMAL)					
CCSID:	(1208)					
Format:	MQSTR					
Flags:	0x00000000					
NameValue:	<mcd><Hds>jms_text</Hds></mcd>					
Len:	32					
Val:	<mcd><Hds>jms_text</Hds></mcd>					

As displayed in the zoomed picture below, our data passes in plain text. We can see the exact message that we got from the application being sniffed by the Wireshark. The contents of the message travelled in plain text and could have been viewed by a third party just as I have done here with Wireshark

```

4d 73 64 3e 6a 6d 73 5f 74 65 78 74 3c 2f 4d 73 Msd>jms_text</Ms
64 3e 3c 2f 6d 63 64 3e 20 20 00 00 00 50 3c 6a d></mcd> ...P<j
6d 73 3e 3c 44 73 74 3e 71 75 65 75 65 3a 2f 2f ms><Dst> queue://
2f 44 45 56 2e 51 55 45 55 45 2e 31 3c 2f 44 73 /DEV.QUE UE.1</Ds
74 3e 3c 54 6d 73 3e 31 37 31 36 39 38 37 33 38 t><Tms>1 71698738
36 32 38 38 3c 2f 54 6d 73 3e 3c 44 6c 76 3e 32 6288</Tm s><Dlv>2
3c 2f 44 6c 76 3e 3c 2f 6a 6d 73 3e 20 20 53 77 </Dlv></ jms> Sw
69 66 74 3a 20 46 4d 42 5a 42 57 47 41 38 ift: FMB ZBWGA8

```

Open the Java code and uncomment line below.

```

//cf.setStringProperty(WMQConstants.WMQ_SSL_CIPHER_SUITE, "*TLS12ORHIGHER");

// Create JMS objects

```

Open the server that is hosting MQ, in this case, it's a Linux server. However, navigate to a directory and execute the command below to create a self-signed cert.

Run the code below to initiate the SSL cert creation process.

```
openssl req -newkey rsa:2048 -nodes -keyout key.key -x509 -days 365 -out key.crt
```












[illegible]

```

+++++
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:BW
State or Province Name (full name) [Some-State]:CETRAL
Locality Name (eg, city) []:Gaborone
Organization Name (eg, company) [Internet Widgits Pty Ltd]:My Company Pty Ltd
Organizational Unit Name (eg, section) []:IT
Common Name (e.g. server FQDN or YOUR name) []:mqs.augcyba.com
Email Address []:lorem@augcyba.com

```

```
root@debian-bullseye-20240519-202906:/home/mmenaysto/store# ls
clientkey.jks  key.crt  key.key
```

 com	2024/05/20 15:04	File folder	
 java_certs	2024/05/30 13:06	File folder	
 clientkey	2024/05/30 12:27	JKS File	2 KB
 clientkey.kdb	2024/05/30 16:05	KDB File	2 KB
 clientkey.sth	2024/05/30 16:05	STH File	1 KB
 com.ibm.mq.allclient-9.3.0.0	2024/05/20 14:57	Executable Jar File	8 145 KB
 javax.jms-api-2.0.1	2024/05/20 15:02	Executable Jar File	63 KB
 json-20220320	2024/05/20 15:03	Executable Jar File	70 KB
 key	2024/05/30 12:28	Security Certificate	2 KB
 key	2024/05/30 12:28	Apple Keynote	2 KB
 mqjms.log.0	2024/05/21 11:37	0 File	1 KB

Now that you have the MQ client libraries, you'll have the MQ security command line tool, *runmqakm*. Enter this command to create a keystore in .kdb format and store the password in a .sth file

```
C:\Users\lorem\MQ\MQClient>runmqakm -keydb -create -db clientkey.kdb -pw [!!pick_a_passwd_here!!] -type pkcs12 -expire 1000 -stash
5724-H72 (C) Copyright IBM Corp. 1994, 2024.
```

After the command is executed we see the newly created password store file.

 clientkey.sth	2024/05/30 16:05	STH File	1 KB
---	------------------	----------	------

Import certificates into the keystore using the command below.

```
C:\Users\lorem\MQ\MQClient>runmqakm -cert -add -label QM1.cert -db clientkey.kdb -stashed -trust enable -file key.crt
5724-H72 (C) Copyright IBM Corp. 1994, 2024.
```

We should have this file in our directory.

 clientkey.kdb	2024/05/30 16:11	KDB File	2 KB
---	------------------	----------	------

When we initialize the container we should provide the directory of the key files.

So from the documentation, it seems we have to create a non running container just for the mere purpose of mounting the files to the storage volume.

```
root@debian-bullseye-20240519-202906:/# docker create --name mq-tls-config-container --mount="type=volume,src=tls-key-vol,dst=/home/mmenayato/store" --user 0:0 icr.io/ibm-messaging/mq:latest
a28cef78732771ee392a3ca667344c677065ac5bcb182444cefa55416c34e398
```

We can see the newly non-running container named *config container*

```
root@debian-bullseye-20240519-202906:/# docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS                               NAMES
sz3ocf787327   icr.io/ibm-messaging/mq:latest      "runmqdevserver"       2 minutes ago Created                                     mq-tls-config-container
```

Copy the files over to the directory in the container.

```
root@debian-bullseye-20240519-202906:/# docker cp /home/mmenaysto/store/. config-container:/etc/mqm/pki/keys/mykey
Successfully copied 8.19kB to config-container:/etc/mqm/pki/keys/mykey
```

This completes the initial set-up. Now, we can enable TLS using the volume created previously and the MQ Docker image.

```
root@mqinstance-20240531-092748:/home/account# docker run -it --name grand_queer --env LICENSE=accept --env MQ_QMGR_NAME=QM1 --mount="type=volum
e,src=tls-key-vol,dst=/etc/mqm/pki/keys/mykey" --publish 1414:1414 --publish 9443:9443 --detach --env MQ_APP_PASSWORD=B2km2bp4A? icr.io/ibm-messa
ging/mq:latest "
```

```
573  chmod o+r /etc/mqm/pki/keys/mykey/key.key
574  chmod o+r /etc/ssl/server/key.*
```

```
DISPLAY CHANNEL(DEV.APP.SVRCONN)
 2 : DISPLAY CHANNEL(DEV.APP.SVRCONN)
AMQ8414I: Display Channel details.
CHANNEL(DEV.APP.SVRCONN)                CHLTYPE(SVRCONN)
ALTDATE(2024-06-03)                     ALTIME(13.34.17)
CERTLABL( )                             COMPHDR(NONE)
COMPMSG(NONE)                           DESCR( )
DISCINT(0)                               HBINT(300)
KAINT(AUTO)                             MAXINST(999999999)
MAXINSTC(999999999)                     MAXMSGL(4194304)
MCAUSER(app)                            MONCHL(QMGR)
RCVDATA( )                              RCVEXIT( )
SCYDATA( )                              SCYEXIT( )
SENDDATA( )                             SENDEXIT( )
SHARECNV(10)                            SSLCAUTH(OPTIONAL)
SSLCIPH(ANY_TLS12_OR_HIGHER)            SSLPEER( )
TRPTYPE(TCP)
```

```
root@mqinstance-20240531-092748:/home/account/mq-secure-ms# docker stop $(docker ps )
4c7b9f2f781d
```