IN MIRTH CONNECT

Best Practices and Vulnerabilities of Mirth Connect

Author:
Jeff Campbell
Technical Consultant,
Galen Healthcare Solutions

Date:
May 15, 2015

galenhealthcare.com

GALEN
HEALTHCARE SOLUTIONS

Table of Contents

Overview

Whether trying to comply with HIPAA, SOX, FIPS or any other federal regulation regarding the robustness and integrity of data, security is a paramount concern when it comes to an interface engine that too often has been underemphasized. When talking about securing an interface engine, most organizations are aware of and take steps to ensure the data entering and exiting the application is secured in some form, usually a VPN as many legacy systems are only capable of traffic over TCP\IP or directly to file. This is often seen as enough as these applications often reside on internal servers where organizations feel they are safe and protected by their own or contracted IT staff and it is only the data leaving the practice that must be secured.

This is not enough in most cases however as there are a myriad of places where sensitive data is left unsecured on the appliance, creating the possibility where a malicious person could breach the integrity of the interface engine. What about the storage of the messages itself as they pass through the application? How is the application for the interface engine accessed and how granular are its permissions? Is there the possibility of intercepting traffic as it leaves the server via a packet sniffer? Many of these areas of concern may have options in the interface engine to be secured, either through global settings on the application or the interface handling the traffic itself, but enabling those options may very well have their own consequences which must be taken into consideration.

As an example of hardening an interface engine and covering many different points of failure for the integrity and security of its messages, I will be highlighting security options for the Mirth Connect interface engine (v3.1.1) in order to promote Healthcare IT best security practices.

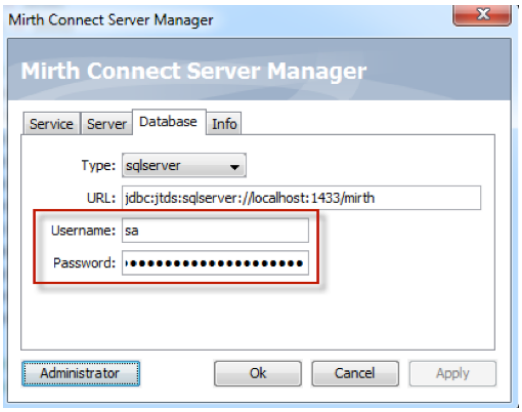
What is Mirth Connect?

Mirth is a cross-platform interface engine that enables bi-directional sending of messages over numerous protocols including TCP/MLLP, directly to database (MySQL, PostgreSQL, Oracle, Microsoft SQL Server, ODBC), file, JMS, FTP/SFTP, HTTP, SOAP, or SMTP.

Securing the Mirth Connect Interface Engine

By way of default installation, there are several areas that are left extremely vulnerable and/or encryption is left either at a lower strength than recommended or disabled entirely. Many of these areas can be configured or added to the mirth.properties file found at the installation path $Mirth/conf/mirth.properties for the Mirth installation. However for most default installations these are not added or configured, allowing for potential vulnerabilities to be left exposed.

For example, the Mirth Connect server service itself connects to a database through the use of connection settings held in the mirth.properties file. From a default installation, even for Mirth-installed appliances, these are often stored in an unencrypted plaintext form, allowing for any and all with access to that file to view potential admin passwords and usernames to the Mirth database. This could potentially expose other databases on the same server as the account the Mirth Connect service is configured to use to may also have access to other databases.




```
65 # examples:
66 #   Derby       jdbc:derby:${dir.appdata}/mirthdb;create=true
67 #   PostgreSQL  jdbc:postgresql://localhost:5432/mirthdb
68 #   MySQL       jdbc:mysql://localhost:3306/mirthdb
69 #   Oracle      jdbc:oracle:thin:@localhost:1521:DB
70 #   SQLServer   jdbc:jtds:sqlserver://localhost:1433/mirthdb
71 database.url = jdbc:jtds:sqlserver://localhost:1433/mirth
72
73 # if using a custom driver, specify it here
74 #database.driver =
75
76 # maximum number of connections allowed for the connection pool
77 database.max-connections = 20
78
79 # database credentials
80 database.username =sa
81 database.password =Im_a_Plaintext_Password
```

Figure 1.1: (Top) Database connections settings in the Mirth Connect Server Manager and (bottom) how they are stored in plaintext on the server.

To correct this vulnerability, an additional encryption setting can be enabled in the mirth.properties file that will prompt the mirth service after a restart to rewrite all plaintext passwords from then on to an encrypted, non-human readable format.

```
32
33 # Enabled encryption of properties with passwords.
34 encryption.properties = 1
35
```

Figure 1.2: After setting the encryption property shown here to 1, all password fields will be encrypted.

```
68 # examples:
69 #   Derby       jdbc:derby:${dir.appdata}/mirthdb;create=true
70 #   PostgreSQL  jdbc:postgresql://localhost:5432/mirthdb
71 #   MySQL       jdbc:mysql://localhost:3306/mirthdb
72 #   Oracle      jdbc:oracle:thin:@localhost:1521:DB
73 #   SQLServer   jdbc:jtds:sqlserver://localhost:1433/mirthdb
74 database.url = jdbc:jtds:sqlserver://localhost:1433/mirth
75
76 # if using a custom driver, specify it here
77 #database.driver =
78
79 # maximum number of connections allowed for the connection pool
80 database.max-connections = 20
81
82 # database credentials
83 database.username =sa
84 database.password ={enc}O8HCUSpgBBesW6gwE+nTa2Q8AUNLoYwChecqkjyWpsM=\r\n
85
```

Figure 1.3: Mirth.properties file after encryption settings have been set. The database.password property is now properly encrypted.

Other Mirth security settings (such as specific cryptographic algorithms, key length, and password controls such as minimum/maximum length, number of special characters, numbers, expiration, etc…) must be enabled or configured from the local mirth.properties file. All of those settings by default are left to the minimum allowable values, which in most cases would not meet most organizations security policies.

Certificates for Mirth Connect Server

It is important for connecting agent, whether they be another interface server or remote application, to be able to verify the integrity of the Mirth server by way of certificate. Depending on your network configuration, this can either be enabled on the appliance itself to present its own certificate or may be enabled on an intermediary device such as a load balancer. If the certificate is configured on the load balancer, with traffic passing back after presentation, configuring it on the Mirth server is only applicable for persons connecting to it internally (by appliance page or otherwise).

If you do need to configure a third party certificate on the Mirth appliance, this can be done from the appliance home page by going to Systems→ Certificates in the menu bar, generating the Certificate Signing Request, and then uploading it to the certificate authority (CA) of your choice.

Once the CA has authorized the certificate and you've purchased the private/public SSL cert, this can be uploaded into the Mirth appliance configuration on the same certificates screen as the CSR was generated.

Optionally, the locally-signed Mirth appliance CA could be installed on agents that needed to connect so it would trust the self-signed cert. However, I would advise this only for connecting locally (for those persons needing to configure and manage the appliance) rather than external organizations as that will likely not pass muster for integrity.

Securing the Mirth Connect Frontend

Users and Permissions

One of the substandard features of Mirth lies in the lack of granularity of permissions for users of the application. When configuring users through a Mirth Appliance, you have the option of provisioning users as a Control Panel User or a Mirth Connect User or both. Permissions administration is limited to those roles; no more permissions beyond that can be set.

A Control Panel user can log into the appliance home page where they can view the current resource statistics, download upgrades, restart services, and otherwise configure the physical or virtual appliance for Mirth itself. A Control Panel user cannot log into Mirth Connect, but they can however access areas of the appliance where they can affect the running operations of the appliance. This role would likely be perfect for an IT staff member whom is likely responsible for ensuring the uptime of the appliance, but not for the interfaces running across it. Depending on your interface setup however, any configuration or restart of services might require close coordination with someone who can directly interact with Channels using the Mirth Connect frontend itself, which leads to Mirth Connect users.

A Mirth Connect user can log into the Mirth Connect application where they can create, edit, view, start, stop, deploy, and undeploy channels (interfaces).They can view the logs of messages coming across each channel and remove or reprocess them. There is no way to assign roles or permissions for a Mirth Connect user beyond assigning a user login, so it is extremely important to ensure that if there is someone who needs read-only access (such as only needing to view or troubleshoot messages) that they are careful about not modifying or changing settings for the interface engine itself or any of the channels as they could inadvertently break functionality of the engine or Channels.

Enforcing Security Policies for Passwords

One of the most common security policies an organization has to enforce surround the passwords used to gain access to sensitive systems. Mirth is fully configurable to enforce these policies by altering the password related properties found in the mirth.properties file. There we can set restrictions on the contents of the password such as a minimum and maximum length, number of special characters and/or numbers, expiration of the password before requiring it to change, number of retries before lockout, etc… Once these properties have changed, it will require a restart of the Mirth Connect service in order for them to take effect.

```
11  # password requirements
12  password.minlength = 0
13  password.minupper = 0
14  password.minlower = 0
15  password.minnumeric = 0
16  password.minspecial = 0
17  password.retrylimit = 0
18  password.lockoutperiod = 0
19  password.expiration = 0
20  password.graceperiod = 0
21  password.reuseperiod = 0
22  password.reuselimit = 0
23
```

Figure 2.1: Password-related configuration settings in the mirth.properties file.

Auditing Mirth Connect Users

In the event of an audit or security inspection, it may be required to validate that user interactions through the Mirth Connect application did not cause a breach. It is possible through the Events view to filter through Mirth Connect user interactions with the interface engine by several different relevant fields.

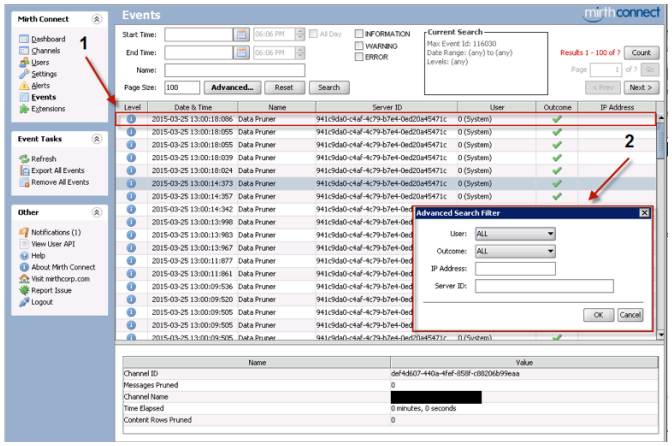


Figure 2.2: The event viewer as seen in the Mirth Connect application. Seen at the top right as indicated by the number 1, various logs for individual user actions are shows in a dashboard list. At the middle-right near the number 2, it is possible to filter these down to a specific user and/or server.

Securing Interfaces in Mirth Connect

While many organizations are concerned with ensuring end-to-end transport of messages between interface engines or other applications are secure, it is just as paramount that any messages or message logs stored by the interface engine are just as secure. For Mirth, many of these encryption or storage settings are controlled on a per Channel basis rather than on a global interface engine-wide one.

For example, to ensure that the message content stored by Mirth is only written in an encrypted manner to the database, you must check the box on the Channel's summary page for "Encrypt Message Content as seen below. This has the double-edged implication of not being able to search free text through the message itself, but most fields you would be interested to troubleshoot on can be placed in alternative locations such as Channel Mappings or Metadata columns where it will be much faster and easier to troubleshoot with.

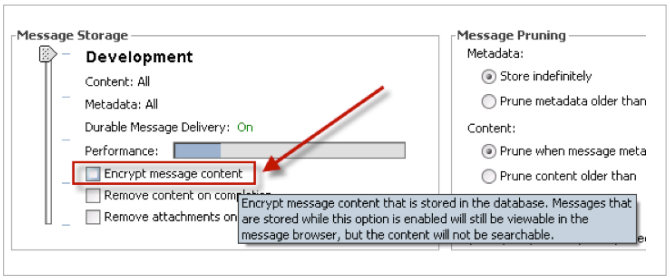


Figure 3.1: Checking the box highlighted in this Channel's summary page will encrypt the message content it stores for it in the logs.

The settings for the encryption algorithm used, key length, and hashing protocol used for the one-way hash can all be configured and set through the mirth.properties file, but by default these settings are not present and must be added and the service restarted to take effect. In the image below, these settings have been configured to:

1. Use the AES encryption protocol.
2. Use a key length of 256 (Maximum for AES)
3. Encrypt anything exported from Mirth such as Mirth Connect server configuration backups, channels, code templates, transformers, and filters.
4. Use the MD5 hashing algorithm for one-way hashing.
5. Use the BouncyCastleProvider as the security provider to use for all encryption and hashing. This is the standard security provider used by Mirth.

```
23
24  # The algorithm to use for encryption (DES, AES, etc.)
25  encryption.algorithm = AES
26
27  # The key length
28  encryption.keylength = 256
29
30  # Enables encryption/decryption of export/import through the Administrator.
31  encryption.export = 1
32
33  # The algorithm to use for one-way hashing (MD5, SHA, etc.)
34  digest.algorithm = MD5
35
36  # The security provider to use for all encryption and hashing.
37  security.provider = org.bouncycastle.jce.provider.BouncyCastleProvider
38
```

Figure 3.2: Encryption settings for Mirth.

SSL Manager (Connectors)

Aside from the certificate presented by the Mirth server, there is also the capability to configure specific channel types to send/receive traffic using SSL/TLS. Currently the connectors available for configuring with the SSL manager are the TCP/IP based connectors; specifically these are the HTTP, Web Services, and File (FTP) connectors. These allow the channels to use the endpoint's SSL certificate to encrypt the traffic with that certificates public key before being sent, and only agents with the private key (typically the destination only) to decrypt the traffic.

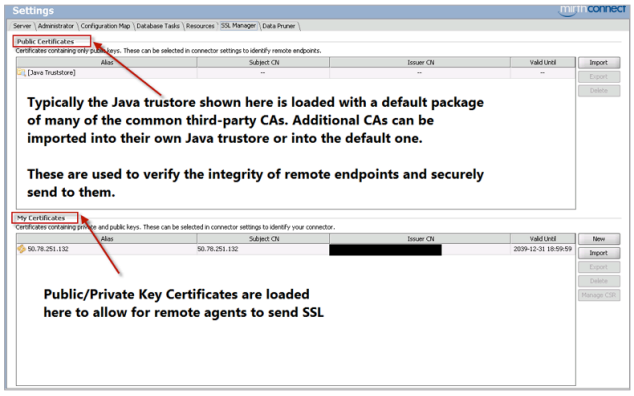


Figure 3.3: Shows the SSL manager's two main areas; the top is where CAs are configured with Mirth for sending SSL and the bottom shows where SSL certificates can be imported for listening connectors using SSL whom need to send securely to them.

As far as configuration goes, Mirth has a local Java truststore (certificate store essentially) where it will use the CAs configured within to determine whether or not a remote destination is trustworthy or not. Depending on the version of Mirth being configured, it is important to note that certificate settings may be handled differently. Prior to 3.1, certificates were configured globally for every SSL connector (so the same Java truststore was automatically used for each connector enabled to use SSL). After 3.1, the connector settings for the SSL manager were all taken out of a centralized area and must be configured on a per channel basis.

For example, for three channels that all use connectors enabled with SSL, prior to 3.1 they would all use the same Java truststore by default. After 3.1, they can use the same truststore, but each channel must be configured to use it manually.

SSL Tunnels

SSL tunnels can also be configured for the Mirth appliance in the event that an SSL connector will not suffice for TCP/IP based traffic or the receiving/sending connector is not supported by the SSL manager natively. SSL tunnels are defined by whether the traffic is inbound or outbound, a receiving port and a sending port (one encrypted and one unencrypted, depending on the direction of the traffic), and the certificates presented by the receiving host.

> "SSL tunnels can also be configured for the Mirth appliance in the event that an SSL connector will not suffice for TCP/IP based traffic or the receiving/sending connector is not supported by the SSL manager natively."

Encrypting Message Content Sent from Mirth

Message traffic sent over SSL to or from Mirth is encrypted as a whole, but were someone to gain access to the private key or be able to intercept the traffic before it passed through an SSL tunnel, the messages would be viewable in plaintext. For example, even if a channel were to send by way of SSL to a remote endpoint, if the traffic itself were decrypted the messages would be human-readable. Another additional security option to overcome this would be to encrypt both the traffic via SSL and the message content itself so that even if the SSL traffic were decrypted, the decrypted traffic would only reveal encrypted message content.

Aside from encrypting the traffic over which the message itself is passing through, Mirth has no capability as of 3.2 for native encryption of messages and this capability must be filled in via third party library or custom programming implemented both in Mirth and the remote interface or application. We have helped implement this for one client for writing encrypted files for temporary hosting/storage where several Federal standards require that data at rest be encrypted to approved secure cryptographic standards. Thankfully, Mirth supports the loading of third-party programming libraries and scripts by way this type of feature can be implemented easily and effectively.

Conclusion

Mirth is a great product for use in Healthcare interoperability and supports a great variety and flexibility of protocols and methods over which to send data, but does require a great deal of configuration to make it fully secure and compliant with Federal requirements.