



TIBCO FTL[®] - Enterprise Edition

Security

Version 7.0.1 | December 2024

Contents

Contents	2
About this Product	4
Security Introduction	5
TIBCO's Security Priority	6
Security Features	8
Security Boundaries	8
Security Vulnerabilities	9
Password Security	9
Authentication	9
TLS	10
Permissions	11
Product Connectivity	12
Developing Secure Applications	14
Ensuring FTL System Security: Tasks for Administrators	16
Enabling TLS for FTL Server	17
Enabling TLS with User-defined Certificates	18
Enabling TLS with FTL-Generated Certificates	19
Enabling TLS for FTL Server	20
Securing Transport Bridges	23
Securing Persistence Services	24
Securing eFTL Services	24
Logging	26

Authentication and Authorization	27
Authentication	28
Using the Built in Flat-File Authentication Service	29
Using the Built in LDAP Authentication Service	31
Using the HTTP/HTTPS Authentication Service	34
Using the External JAAS Authentication Service	34
Using the external custom HTTP / HTTPS based authentication service	36
Using the Built in mTLS Based Authentication Service	36
Using the built in OAuth 2.0 based authentication service	38
Single Sign-On with OAuth 2.0	41
Authenticating to FTL Server	41
Authenticating with Basic Authentication	42
Authenticating with mTLS	42
Authenticating with OAuth 2.0	43
Authorization	45
FTL Server Authorization Groups	45
Mapping Authorization Groups	46
Permissions	47
Configuring Permissions	50
Required Permissions for API Calls	52
Migrating to FTL 6.8.0 when Using Permissions	55
TIBCO Documentation and Support Services	56
Legal and Third-Party Notices	58

About this Product

TIBCO® is proud to announce the latest release of TIBCO FTL® software.

This release is the latest in a long history of TIBCO products that use the power of Information Bus® technology to enable truly event-driven IT environments. TIBCO FTL software is part of TIBCO Messaging®. To find out more about TIBCO Messaging software and other TIBCO products, please visit us at www.tibco.com.

Security Introduction

At TIBCO, security is our highest priority. You can be sure that TIBCO FTL messaging is secure based on our own high standards and the best practices of our infrastructure providers.

FTL offers logging, transport security, authentication and authorization, and compatibility with other products.

This document describes procedures to ensure security within FTL components and the communication between components. It also provides security-related guidance for other aspects of internal and external communication including product connectivity and configuration of security options.

This document is useful to security officers, deployers, administrators, and purchasers. Developers may be interested in the Coordination Forms section.

This section is an overview of FTL security features. The [FTL documentation](#) also includes security recommendations for fine tuning the security of FTL as well as security tips for Developers.

TIBCO's Security Priority

At TIBCO, security is our highest priority. TIBCO maintains a company-wide information security management system and control program. This includes security policies, standards, and procedures based on [ISO/IEC 27001:2013](#).

TIBCO incorporates the STRIDE model to help analyze and find threats to ensure system and service integrity and reliability in processes, data stores, data flows, and trust boundaries. Our approaches ensure that applications and systems fulfill the CIA triad (confidentiality, integrity, and availability).

[STRIDE and TIBCO Assurance]

Threat	TIBCO Assurance
Spoofing identity	Authenticity - authentication
Tampering with data	Integrity and auditing
Repudiation threats	Non-repudiability
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

TIBCO adheres to privacy and security requirements related to the protection and processing of individual personal data (collectively, the "Protected Data"). For details, see our [Customer Privacy and Security Statement](#).

TIBCO has adopted policies and practices in alignment with industry best practices. This includes quickly addressing and disclosing vulnerabilities. TIBCO has an incident response policy and plan. The policy ensures that security incidents are identified, contained, investigated, and remedied. For details or to report a potential security issue, see security@TIBCO.

TIBCO has policies that guide our Product Life Cycle (PLC). The policies include peer reviews, static code analysis, and both manual and automated QA processes. In addition,

we routinely run performance testing on any new or updated software to ensure the highest quality. There is a clear division between devops and development. All changes are logged in our source code repository. We use standard deployment tools. Our iterative methodology ensures a functional view of the processes, milestones, activities, and artifacts, or records.

TIBCO's Business Continuity Plan (BCP)

TIBCO has a Business Continuity Plan (BCP) to ensure that the effects of an emergency event are minimized. If a disaster or emergency situation occurs, the Support Emergency Team (SEMT) coordinates the recovery effort and uses the Employee Communication Chains to notify staff that the BCP is activated.

TIBCO has an Information Security Management System (ISMS) to preserve the confidentiality, integrity, and availability of information. Information security is considered in the design of processes, information systems, and controls.

TIBCO's Quality Management System (QMS) is based upon ISO 9001. ISO 9001 is an internationally recognized standard that sets out the criteria for a quality management system incorporating the Plan-Do-Check-Act (PDCA) cycle. TIBCO's QMS is a formalized system of business processes, procedures, and responsibilities focused on

- Customer Excellence - Meeting customer requirements and enhancing customer satisfaction by providing high-quality products and services
- Quality - Meeting TIBCO requirements for quality policies and objectives with measurable goals

TIBCO's Quality Management System is documented and structured in levels.

Security Features

TIBCO FTL software includes the following security features:

- The choice of user-defined TLS certificates or TLS certificates generated by FTL.
- TLS for communication among applications and FTL servers.
- HTTPS for secure access to the administrative user interface and REST API.
- A choice of various built-in and customizable authentication mechanisms.
- Coarse separation of users into three categories, application clients, servers, and administrators.
- Fine-grained permissions for persistence clusters and eFTL channels.
- Full configuration control
- Monitoring and logging

Security Boundaries

FTL logs key activities. Your organization is responsible for protecting and reviewing those files.

FTL ensures that administrative interfaces properly authenticate and authorize users. Your organization is responsible for configuring permissions for those users.

Your organization is responsible for making sure that the running engines have sufficient resources to handle the loads created by the applications.

The [OWASP Top 10](#) document and the [CWE Top 25 Most Dangerous Software Weaknesses](#) can help your organization ensure that the most critical security risks are covered.

Security Vulnerabilities

Security features that protect FTL connections and communications depend on the implementation of OpenSSL for implementation Transport Layer Security (TLS) protocols. If the security of OpenSSL were compromised, FTL and applications that use FTL could be vulnerable as well.

For information about OpenSSL configuration, components, and downloads, see [OpenSSL.org](https://www.openssl.org).

In addition to the key security technologies described, security depends on your organization correctly configuring and using FTL's components and capabilities.

Password Security

Keystore passwords encrypt key files, such as the private key file that FTL servers use to identify themselves to clients and to other servers.

Passwords can be masked. You can mask passwords using `tibftladmin`. Masked passwords have `$mask$` at the beginning of the string. Masked passwords are unmasked before being sent to the realm service.

FTL allows you to supply a password in several ways based on the level of security desired.

For details, see "Password Security", "Security: Clients" section in [TIBCO Administration](#).

Authentication

Administrators can configure the FTL server to require authentication for connections from clients and other FTL servers. When authentication is enabled, administrators can optionally enable authentication for specific FTL services, such as the persistence service or the eFTL service, or for peer-to-peer FTL transports. For details, see “Securing Persistence Services” and “Securing eFTL Services”.

FTL server can be configured to use one or more of the following authentication providers:

- Built-in “flat-file” authentication
- Built-in ldap authentication

- Built-in oauth2 authentication
- Built-in mTLS authentication (also requires TLS)
- Customizable authentication to a user-provided HTTP(S) service (for example, a user-provided service that implements JAAS)

For details, see [Authentication](#)

At a minimum, the administrator must assign users to one of three categories and configure the authentication provider accordingly:

- Clients have the “ftl” role, which allows them to connect to FTL server or other FTL clients.
- Servers have the “ftl-internal” role, which identifies them as fully privileged servers when connecting to other FTL servers.
- Administrators have the “ftl-admin” role, which identifies them as administrators to the FTL server for the purpose of monitoring and making configuration changes.

For details, see [FTL Server Authorization Groups](#)

TLS

Administrators can configure the FTL server to require TLS for connections from clients and other FTL servers. When enabled, any connection that requires authentication will also require TLS. Authentication is always required to use TLS. See [Authentication](#) for an overview.

Administrators can configure TLS in one of two ways:

- FTL-generated certificates: TLS certificates are generated by FTL server. This is required when enabling TLS and authentication for peer-to-peer FTL transports or transport bridges.
- User-defined certificates: TLS certificates are provided by the administrator. This allows the administrator to have full control over the use of TLS certificates in FTL. This is required for mTLS authentication.

For details, see [Enabling TLS for FTL Server](#)

In addition, for situations where TLS termination occurs at an ingress point (before the connection reaches FTL server), it is possible to enable TLS only at the client. In this case,

for all connections that require authentication, the client will also use TLS. For these same connections, FTL server will require authentication, but not TLS.

Permissions

In addition to authentication, administrators can additionally enable fine-grained permissions to control which users can send or receive messages. Various permissions may be configured for persistence clusters, persistence stores, or eFTL channels. For details, see [Authorization](#)

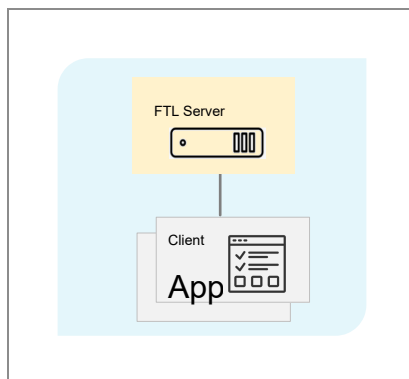
To use permissions, authentication must be enabled for FTL server and the affected persistence services or eFTL services. TLS may optionally be enabled at FTL server and the client, or just at the client.

Product Connectivity

TIBCO FTL includes several interconnecting components, and also connects with other TIBCO and third-party products. You can secure all connections within the TIBCO product family.

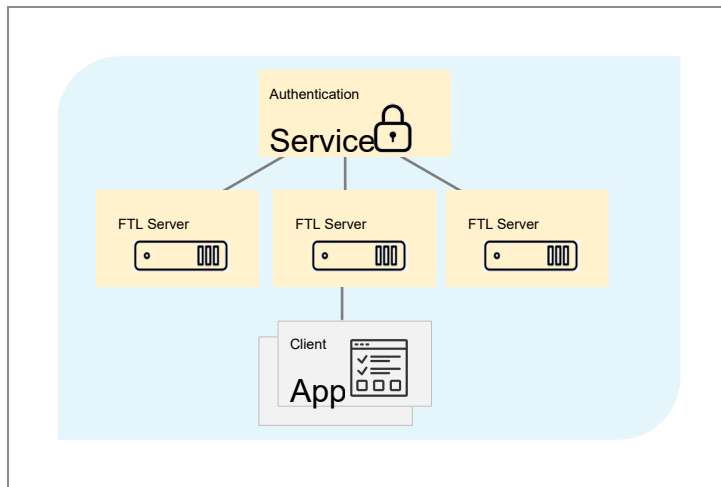
The following diagram, [Development Minimum Deployment](#), illustrates that a development environment only requires an FTL server and a client app.

Figure 1: Development Minimum Deployment



The following diagram, [Production Minimum Viable Secure Deployment](#), illustrates that a production environment must minimally have three FTL servers, an authentication service communicating with each FTL server, and a client application. You must secure these components.

Figure 2: Production Minimum Viable Secure Deployment

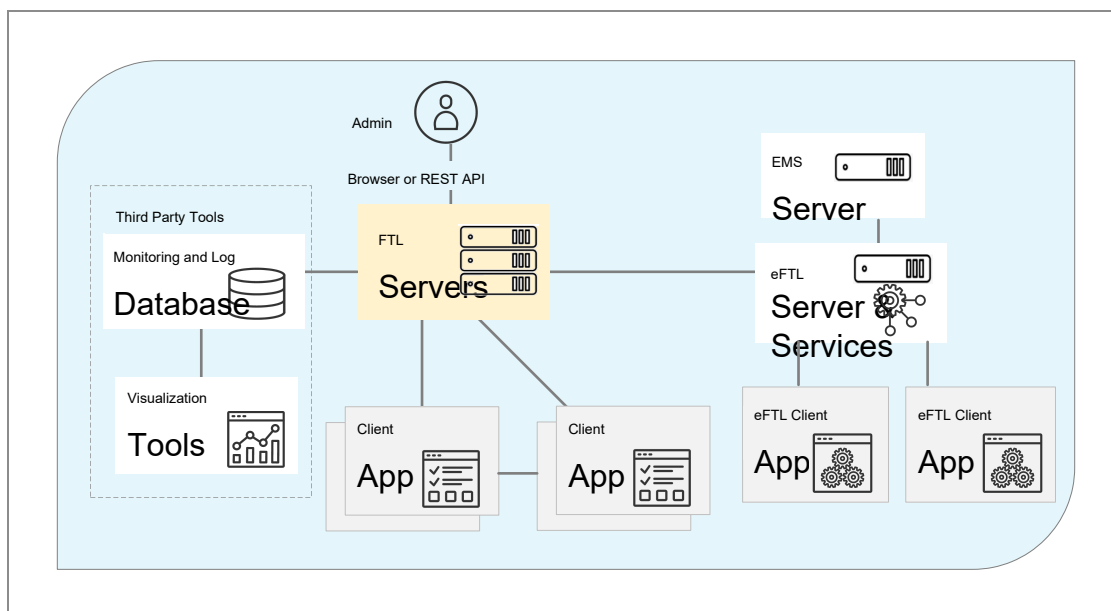


The following diagram, [Components Connecting in the FTL Realm](#), depicts other components that may connect to your FTL realm.



Caution: FTL components attempt to interact with third-party components in a secure fashion. TIBCO does not warrant the security of third-party products.

Figure 3: Components Connecting in the FTL Realm



Developing Secure Applications

To implement security, application developers focus on the realm connect call and its arguments. Complete this task, or use its steps as a checklist.

Administrators determine whether authentication and, optionally, TLS are required for client connections.

If authentication is required, application developers must pass properties that identify the application to the realm connect call. There are three possibilities:

- **Basic auth (username and password):** The application must set the user name and user password properties. The administrator must configure at least one of these auth providers like the built-in flat-file authenticator, the built-in ldap authenticator, or the customizable HTTP(s) authenticator.
- **mTLS auth:** The application must set the `CLIENT_CERT` and `CLIENT_PRIVATE_KEY` properties. If the private key is encrypted, the application must set the `CLIENT_PRIVATE_KEY_PASSWORD` property. TLS is required (see below). The administrator must configure FTL server to use the mTLS auth provider.
- **Oauth2:** The application must provide an oauth2 access token (in signed JWT format), or the URL of an oauth2 server and the credentials needed to obtain an access token. The administrator must configure FTL server to use the oauth2 provider.

If TLS is required (in addition to authentication), application developers must specify the https scheme for all URLs passed to the realm connect call. Applications must also specify how to trust FTL server's TLS certificate. There are two possibilities:

- **Trust file:** The application must set the `TRUST_FILE` property. The trust file is supplied by the administrator. This is the only valid method when the administrator uses FTL-generated certificates.
- **System trust store:** If the application does not set a trust file, the FTL library loads the system trust store. The trust roots for the certificate used by the administrator must be installed in the system trust store. This method is only supported when the administrator configures user-defined certificates. (The FTL-generated trust file cannot be installed in the system trust store.)

For details, see [Enabling TLS for FTL Server](#)

When using TLS with user-defined certificates, the host name passed to the realm connect call must match the subject alternative name in FTL server's certificate, or the connection fails.

When using TLS, FTL application developers may specify the openssl security level. (The openssl library is used for connections to FTL server or other FTL clients.) In general, a higher security level requires stronger certificates and encryption.

For details, see the developer [API Documentation](#) in Web Help or in the FTL include directory.

After the realm connect call, additional secure transports used by the application will automatically enforce authentication and TLS requirements. The administrator must decide which transports to secure (for example, persistence service transports, eFTL service transports, group service transports, or peer-to-peer transports).

The administrator may optionally configure additional authorization checks (permissions) for persistence stores and eFTL channels.

For details, see [Authorization](#)

Ensuring FTL System Security: Tasks for Administrators

To ensure security among FTL applications, eFTL applications, FTL servers, and administrative tools, administrators complete the following tasks.

Procedure

1. Applications:

You must enable authentication and authorization at FTL server. If applications need to communicate directly with each other, then you must enable TLS using FTL-generated certificates. See [Enabling TLS for FTL Server](#). Also, all transports used for peer-to-peer communication must be marked as secure in the realm configuration. Use only these transport protocols.

- Secure Dynamic TCP
- Secure TCP

If applications communicate with services provided by FTL server (for example, persistence services), then you may use TLS with FTL-generated certificates or user-defined certificates. Then take additional steps:

- TLS may be enabled at FTL server (see step 3). Alternatively, administrators may provide TLS termination at an ingress point, or secure the network through other means.
- Authentication and authorization must be enabled for all relevant services (see step 4).

2. Authentication and Authorization:

- a. Configure authentication and authorization.
 - i. Your role includes configuring your enterprise authentication and authorization system (such as an LDAP service) with appropriate information to support TIBCO FTL components and application users.
 - ii. For details, see [Authentication](#).

3. Enabling TLS:

- a. When enabling TLS at FTL server, you must choose whether to use FTL-generated certificates or user-defined certificates. For details, see “Enabling TLS for FTL Server”.
- b. If providing TLS termination at an ingress point with a user-defined certificate, do not enable TLS at FTL server. Instead, instruct application developers to enable TLS in their applications, and provide the appropriate trust file.

4. TIBCO FTL Component Services:

- Secure all transport bridges. Verify that the transports interconnected by the bridges use only secure transport protocols.

For details, see [Securing Transport Bridges](#).

- Secure all persistence services. Configure the persistence clusters so that all relevant transports use only secure transport protocols.

For details, see [Securing Persistence Services](#).

- Secure all eFTL services.

TIBCO eFTL services must use secure transports to communicate with one another, and with eFTL applications. Your role includes these subtasks:

- Reconfigure the automatically-generated eFTL transport definitions so that all relevant transports use only secure transport protocols.
- Configure channels with appropriate authorization groups.
- Coordinate with application developers to ensure that eFTL clients connect to the eFTL services using the secure web sockets protocol (WSS).

5. Secure the FTL server data directories and files against unwanted access by other users.

Enabling TLS for FTL Server

FTL supports two different mechanisms for enabling TLS at FTL server:

- User-defined certificates, where administrators provide TLS certificates to FTL server (and possibly clients, for mTLS authentication). The administrator is responsible for

choosing a certificate authority and obtaining the certificates. FTL server loads its certificate on startup.

- FTL-generated certificates, where administrators run FTL tools to initialize TLS keys and trust roots for an FTL realm. At runtime, FTL server issues TLS certificates as needed.

FTL and eFTL components can use TLS 1.2 or TLS 1.3. TLS 1.3 is used wherever possible.

The openssl library is used for FTL transports between FTL components (on both client and server side). The openssl library is also used for the server side of any HTTPS and eFTL client connections accepted by FTL server.

Administrators may configure the openssl security level. In general, a higher security level requires stronger certificates and encryption. See [FTL Server Configuration Parameters](#)

Enabling TLS with User-defined Certificates

To enable TLS with user-defined certificates, specify the following parameters in the FTL server yaml configuration file:

- `tls.server.cert`
- `tls.server.private.key`
- `tls.server.private.key.password` (if needed)
- `tls.client.trust.file` (if needed)

For details, see [FTL Server Configuration Parameters](#)

On starting, FTL server loads the certificate and private key. The private key password is used to decrypt the private key. When making outgoing TLS connections to other FTL servers, FTL server verifies the remote server's certificate using the trust file, or the system trust store if no trust file is specified.

When clients and other FTL servers connect to an FTL server, they ensure that the hostname used to connect to FTL server matches the subject alternative name (SAN) in FTL server's certificate. The certificate must have a SAN. In the certificate's SAN, the leftmost component of a DNS name may be a wildcard.

i Note: The certificate must look valid to both clients and FTL servers. If clients and servers use different hostnames to connect to FTL server, the certificate's SAN must contain multiple entries and/or a wildcard entry.

If you are terminating TLS at an ingress point for client connections, do not configure FTL server with a TLS certificate. This means that FTL server does not use TLS for communication with clients or other FTL servers.

i Note: Secure peer-to-peer transports (used for direct communication between applications) are not permitted when user-defined certificates are configured. If applications must communicate directly (without FTL server in the middle), see [Enabling TLS for FTL Server](#)

See `samples/yaml/tls-user` for details on how to setup FTL Server with user-defined certificates.

Enabling TLS with FTL-Generated Certificates

Instead of taking a certificate from the user, FTL server can generate and manage TLS certificates. This is required in the following cases:

- Applications communicate directly with each other, using secure peer-to-peer transports.
- Applications communicate with each other via transport bridges, using secure transports

Before you begin

- Choose a keystore file password, and determine the appropriate level of security for that password.
 - Ensure that the clocks on all servers in a cluster are synchronized.
1. Remove any obsolete TLS data files from the FTL servers' data directories.
 2. Generate TLS data files.

To generate full-security files, enter:

```
tibftlserver --init-security file:<pw_file_name> -c <my_config_file_path>
-n <svr_name>
```

This command instructs the FTL server to generate new TLS data files, encrypting the new keystore file with the password.

(If the FTL server detects existing TLS files, it does not generate them anew. However, the FTL server does not decrypt or inspect existing files.)The server generates TLS files in the data directory (specified in the configuration file). If the data directory is unavailable, the server writes these files to the current directory. After writing the files, the FTL server exits.

3. Distribute the TLS files.The keystore file and trust file must be distributed to all FTL servers which include all core servers and auxiliary servers at all sites (including primary, satellite, and DR sites).Every server uses the same private key to identify itself. Every server uses the same trust file to verify the identity of FTL servers.
 - Supply copies of the keystore file and trust file to every FTL server. Place these files in the data directory of the servers.



Note: Specify the data directory in the configuration file for each FTL server

- Supply a copy of the trust file to every client including application programs and browsers that access the FTL server GUI.



Note: When a server generates new TLS data files, you must redistribute these files.



Note: See the ftlstart script in the samples directory. The --secure option illustrates a basic way to start a secure FTL server.

Enabling TLS for FTL Server

Secure FTL servers are central to the security of any enterprise that communicates using TIBCO FTL messaging software. To secure the FTL servers, complete this task. An FTL

server can generate all the data it requires for TLS, except for the keystore password, which you must supply.

The FTL server supports TLS 1.3. FTL components use TLS 1.3 when communicating with each other. If possible, the FTL server uses TLS 1.3 when communicating with other components, such as a browser or an eFTL client.

Before you begin

- Secure the FTL server data directories and files against unwanted access by other users.
- The enterprise authentication system (for example, and LDAP system) must define usernames and associate them with appropriate FTL authorization groups.
- An authentication service (either internal or external) must be running. For background information, see [Authentication Service](#) in TIBCO FTL® - Enterprise Edition [Administration](#).
- Choose a keystore file password, and determine the appropriate level of security for that password.
- Ensure that the clocks on all servers in a cluster are synchronized.

Procedure

1. Remove any obsolete TLS data files from the FTL servers' data directories.
2. Generate TLS data files.

To generate full-security files, enter:

```
tibftlserver --init-security file:<pw_file_name> -c <my_config_
file_path> -n <svr_name>
```

To prepare the server for authentication-only operation, enter:

```
tibftlserver --init-auth-only
```

This command instructs the FTL server to generate new TLS data files, encrypting the new keystore file with the password.

(If the FTL server detects existing TLS files, it does not generate them anew. However, the FTL server does not decrypt or inspect existing files.)

The server generates TLS files in the data directory (specified in the configuration file). If the data directory is unavailable, the server writes these files to the current directory. After writing the files, the FTL server exits.

3. Distribute the TLS files.

The keystore file and trust file must be distributed to all FTL servers which include all core servers and auxiliary servers at all sites (including primary, satellite, and DR sites).

Every server uses the same private key to identify itself. Every server uses the same trust file to verify the identity of FTL servers.

- a. Supply copies of the keystore file and trust file to every FTL server.

Place these files in the data directory of the servers.



Note: Specify the data directory in the configuration file for each FTL server.

- b. Supply a copy of the trust file to every client including application programs and browsers that access the FTL server GUI.

For more information, see [Trust File](#) in TIBCO FTL® - Enterprise Edition [Administration](#).



Note: When a server generates *new* TLS data files, you must redistribute these files.

4. Configure the FTL servers to use TLS security and supply the keystore file password as the property value:

```
globals:
  # ...
  tls.secure: <password_argument>
```

FTL servers use the password to encrypt and decrypt the keystore file. For information on the form of the password argument, see [Password Security](#) in TIBCO FTL® - Enterprise Edition [Administration](#).

5. Configure the FTL server properties related to the authentication service.

FTL servers authenticate and authorize client credentials using the authentication service. Configure the authentication service in the FTL server configuration file.

6. Configure the username and password for communication with affiliated FTL servers. If satellite or DR FTL servers are used, the primary FTL servers must authenticate themselves to the satellite or DR servers and vice versa. Add an appropriate username and password to the configuration file of all FTL servers (primary or satellite or DR). Ensure that this user has the `ftl-internal` role. See [FTL Server Configuration Parameters](#), "Affiliated FTL Servers".

```
globals:
  # ...
  user: <username>
  password: <password_argument>
```

7. Start the FTL server processes.

Start servers using a standard command line (that is, without the `--init-security` option). For example:

```
tibftlserver -c <config_file> -n <server_name>
```



Note: See the `ftlstart` script in the `samples` directory. The `--secure` option illustrates a basic way to start a secure FTL server.

Securing Transport Bridges

To secure a transport bridge, complete this task.

Before you begin

All FTL servers must enable authentication. All FTL servers must enable TLS using FTL-generated certificates.

Procedure

1. Verify that the transports interconnect by the bridge. Configure only secure network transport protocols. This ensures that authentication and TLS are enabled for all

connections.

Use only these transport protocols:

- Secure Dynamic TCP
- Secure TCP

Securing Persistence Services

To secure a persistence service, complete this task.

Before you begin

All FTL servers must enable authentication. Administrators may optionally enable TLS and fine-grained permissions.

1. Verify that the persistence cluster definition specifies secure transport protocols. This ensures that authentication (and, optionally, TLS) is enabled for all connections. The client protocol, cluster protocol, disaster recovery (DR) protocol, and inter-cluster protocol must be secure.

Use only these transport protocols:

- Secure Dynamic TCP
 - Secure TCP
 - Secure Auto
2. For further details, see "Clusters Grid" in TIBCO FTL [Administration](#)
 3. For details, see [Authorization](#)

Securing eFTL Services

To secure an eFTL service, complete this task.

Before you begin

All FTL servers must enable authentication. Administrators may optionally enable TLS and

fine-grained permissions.

If any channels use EMS servers or FTL persistence services, those services must also be secure.

Procedure

1. Verify secure transport protocols.

This ensures that authentication (and, optionally, TLS) is enabled for all FTL transport connections.

The cluster-facing transport and all the channel application-facing transports must be secure. Check their protocols in the transports grid

Use only these transport protocols:

- Secure Dynamic TCP
- Secure TCP
- Secure Auto

2. Include authenticated usernames.

Specify the parameter `publish.user` in the eFTL service section of the FTL server configuration file.

With this option, the eFTL service appends a field to messages published by eFTL client apps when it forwards them to FTL and EMS subscribers. That field contains the authenticated username of the eFTL publisher. FTL and EMS application code can use this username to authorize requests.

3. Enable authentication for eFTL client connections.

4. Optional. Enable fine-grained permissions for eFTL channels.

5. Optional. Enable TLS for FTL server. This will also enable TLS for eFTL client connections. For details, see [Enabling TLS for FTL Server](#)

6. [Enabling TLS for FTL Server](#)

[Enabling TLS for FTL Server](#)

7. For details about the content of that file, see [SSL Parameters for EMS Connections](#) in [TIBCO eFTL Administration](#).

Logging

FTL server has the ability to log events related to security. The loglevel should be specified in the `ftlserver.properties` section of the yaml configuration file. For the configuration reference, see "loglevel" in [FTL Server Configuration Parameters](#)

To log authentication results for incoming connections from clients or other FTL servers, set the loglevel to `auth:verbose`.

To log TLS configuration parameters and the establishment of TLS connections, set the loglevel to `tls:verbose`.

Logging components may be combined with a semicolon as delimiter.

For example, the loglevel can be set to `"auth:verbose;tls:verbose"`.

Elevated logging is not recommended for deployments with many short-lived, recurring connections. The best practice is to develop applications with long-lived connections to FTL server.

Authentication and Authorization

Authentication in FTL is enabled by specifying one or more authentication providers through the `auth.providers` field of the FTL server yaml configuration. For the configuration reference, see the [FTL Server Configuration Parameters](#) and For more details, see [Authentication](#)

When `auth.providers` is set, authentication is required for the following interfaces:

- Realm connections made through the FTL client API.
- The FTL server user interface.
- The FTL server REST API.

FTL offers optional features in addition to the above. To enable authentication for these features, do the following:

- Peer-to-peer FTL transports: Ensure that all transports used for direct communication between FTL clients are secure. Use transport protocol `Secure Dynamic TCP` or `Secure Static TCP`. For more information, see [Transports Grid](#)

i Note: TLS with FTL-generated certificates is required for secure peer-to-peer transports. For more information, see [Enabling TLS for FTL Server](#)

- Transport bridges: Follow the steps for securing peer-to-peer FTL transports.
- Persistence services: All transports used by the persistence cluster must use transport protocol `Secure Dynamic TCP`, `Secure Static TCP`, or `Secure Auto`. TLS is optional.
- Group service: All group service transports must use protocol `Secure Auto` or `Secure Dynamic TCP`.
- Eftl channels: Ensure that all transports used by the eftl cluster use protocol "Secure Auto" or "Secure Dynamic TCP". Ensure that authentication is enabled for the eftl cluster. See "**eFTL Clusters Grid**", in **eftl administration**].

In addition to successfully authenticating, users must belong to specific authorization groups (or roles) to access these interfaces. For more details, see [FTL Server Authorization Groups](#).

Administrators may optionally configure fine-grained permissions for the following features:

- Persistence services: See [Authorization](#)
- Eftl channels: See client Authentication and Authorization in eftl administration

Authentication

When authentication is enabled, FTL clients, eFTL clients, administrative tools, and other FTL servers must authenticate to the FTL server. They can authenticate to the FTL server in one of three ways:

- **Basic authentication:** The user provides a username and password.
- **mTLS authentication:** The user provides a TLS certificate and its corresponding private key. The FTL server verifies the client's certificate during the TLS handshake.



Note: mTLS authentication is not supported for eFTL clients or the eFTL REST API.

- **oauth2 authentication:** The user provides a signed JWT token, or the URL of an oauth2 server that can issue a signed JWT token and also provide credentials for accessing that server.

For more details, see [Authentication](#)

FTL server supports various authentication providers. Each authentication provider has its own configuration and is used for exactly one of the authentication modes above (basic, mTLS or oauth2). The purpose of the authentication provider is to determine if the client has authenticated successfully and if authenticated to determine the client's username and authorization roles. In the case of basic authentication, FTL server can try each basic authentication provider until one succeeds, or they all fail.

The following are the supported authentication providers. More than one provider can be configured. However, duplicate providers are not allowed. For example, it is illegal to configure multiple flat file authentication providers, but it is legal to configure a flat file provider and an ldap provider.

- **Flat File: Basic authentication only:** A built-in provider within FTL server that reads

a file of usernames and passwords. For details, see [Using the Built in Flat-File Authentication Service](#)

- **LDAP:Basic authentication only.** A built-in provider within FTL server that uses an LDAP server to authenticate clients. For details, see [Using the Built in LDAP Authentication Service](#)
- **HTTP/HTTPS (customizable):** Basic authentication only. FTL server makes HTTP/HTTPS calls to an external authentication service to authenticate clients. The authentication service may be completely customized by the administrator. For details, see [Using the external custom HTTP / HTTPS based authentication service](#)
- **mTLS: mTLS authentication only:** Enables verification of client certificates during a TLS handshake. For details, see [Using the Built in mTLS Based Authentication Service](#)
- **oauth2: oauth2 authentication only:** Enables verification of signed JWT tokens, plus enforcement of the token's expiration time under certain circumstances. For details, see [Using the built in OAuth 2.0 based authentication service](#)

Using the Built in Flat-File Authentication Service

The FTL flat-file authentication service provides authentication functionality for the FTL server, reading username and password data from a flat file. It runs inside the FTL server. Set **auth.providers to file:<file-path>** in order to use the built-in flat-file authentication service. For details see, [FTL Server Configuration Parameters](#)

Procedure

1. Configure the flat file with username and password data.

Passwords must be clear text, not obfuscated nor checksummed, but they can be hashed.

i Note: Syntax Summary for flat file

- Each line defines one user.
- Each line must specify a username and password, and may also specify optional authorization roles or groups.
- Delimit the username with a required colon.
- You may add optional space characters after the colon. The password begins with the first non-whitespace character after the colon.
- Delimit the password with a comma-space pair. If a line contains more than one comma-space pair, the *rightmost* pair delimits the password. Earlier pairs become part of the password, as do individual comma and space characters.
- Hashed passwords are allowed.
- Separate authorization roles or groups with a comma *only* (spaces are not valid).

For example:

```
admin: my_admin_pw, ftl,ftl-admin
ftl_svr: my_ftl_svr_pw, ftl-internal,ftl-admin,ftl,auth
app_user_1:my_pw, ftl
app_user_2:      her_pw, ftl
app_user_3:  my pw, more pw,, and still more pw , role-1,ftl
```

In the last example, the boldface type illustrates a complicated password containing spaces, commas, and even comma-space pairs.

2. Supply the location of the flat file through the configuration parameter `auth.providers`

For example:

```
auth.providers: file:/opt/tibco/ftl/samples/yaml/basic-auth/users.txt
```

See also: `samples/yaml/basic-auth/tibftlserver_basic_auth.yaml` in the FTL installation.

Using the Built in LDAP Authentication Service

When LDAP authentication is enabled, FTL server delegates authentication requests to the LDAP server.

To enable LDAP authentication, ensure that the "auth.providers" parameter in the FTL server configuration file contains an LDAP URL. The URL can be either `ldap://<host>:<port>`, or `ldaps://<host>:<port>`.

In addition, configure the `ldap.config` parameter in the FTL server yaml configuration file. This file contains configuration for connecting to the LDAP server.

For details, see [FTL Server Configuration Parameters](#).

For an example, see `samples/yaml/ldap` in the FTL installation directory.

Table here shows various ldap authentication service related parameters to be specified in the file specified in **ldap.config**

LDAP Configuration name	Type	Examples	Description
ldap.user.basedn	String	ou=People,dc=ftl	Based DN used to search for ldap users
ldap.user.scope	String	ldap_user_scope	<p>The scope of the search. Valid values include:</p> <ul style="list-style-type: none">• onelevel• subtree• object <p>The default is to use a one level search.</p>
ldap.user.class	String	ldap_user_class	Criteria used for user search. What class indicates a

			user, what attribute contains a user's unique identifier Unless ldap.group.filter is specified, default search filter is auto generated from relevant configuration parameters
ldap.user.attribute	String	uid	The attribute that is compared to the user name for the search. The default is uid.
ldap.user.filter	String		<p>The filter used when searching for a user.</p> <p>If a more complex filter is needed, use this property to override the default.</p> <p>Any occurrence of {0} in the search string is the user attribute, and {1} is replaced with the user name.</p> <p>The default is {0}={1}</p>
ldap.group.basedn	String	ou=Groups,dc=ftl	The base path for

			the LDAP static group search. If null or not set, static groups are not searched.
ldap.group.scope	String	subtree	<p>The scope of the static group search. Valid values include onelevel, subtree, and object.</p> <p>Default is to use a subtree search.</p>
ldap.group.filter	String		Criteria used for static group search, similar to that used for user search Unless ldap.group.filter is specified, default search filter is autogenerated from relevant configuration parameters
ldap.group.static.class	String	groupofuniqueNames	
ldap.group.static.attribute	String	cn	The attribute of a static LDAP group that contains the group name. Default is cn.
ldap.group.static.member.attribute	String	uniqueMember	The attribute ID of a dynamic

LDAP group object that specifies the name of members of the group. Default is uniqueMember.

Using the HTTP/HTTPS Authentication Service

You can write the http/https based authentication service similar to JAAS based Authentication service.

Using the External JAAS Authentication Service

The FTL JAAS authentication service is a containerized service that provides JAAS functionality for the FTL server. The service connects to your enterprise LDAP server.

If you use an authentication service that is external to the FTL server, you must start it *before* starting the FTL server.

These steps start the sample external JAAS authentication service in a Jetty container. To modify the sample service, see the file `<TIBCO_HOME>/ftl/<version>/samples/jaas/FTL-JAAS.readme.md`.

1. Configure the JAAS file.

The JAAS file specifies Java classes that implement authentication. (You can reuse the same JAAS file as you used in Release 5.x and earlier.)

For details, see *JAAS Login Configuration File* in Oracle Java documentation.

2. Deploy the service in a Jetty application container. The WAR file `FTL-JAAS.war` implements the service. Copy the example WAR file from `<TIBCO_HOME>/ftl/<version>/samples/jaas/FTL-JAAS.war` into Jetty's `demo-base/webapps`

directory.

3. Configure the JAAS realm within Jetty.

Edit `demo-base/etc/login.conf` and add a `tibftlserver` section. See the sample file `<TIBCO_HOME>/ftl/<version>/samples/jaas/ldap.jaas` for an example.

Note: The term "realm" denotes two separate concepts in JAAS and FTL.

4. Optional. Configure TLS within Jetty.

It is good practice to use TLS security for authentication service communications.

5. Start the authentication service in a Jetty container.

Clients can reach the authentication service at `<protocol>://<host>:<port>/FTL-JAAS/login`. (Supply this URL to the FTL server in the next step.)

6. Configure the FTL server to use the authentication service.

The FTL server is a client of the authentication service.

Supply the `yaml` parameter `auth.providers` to specify the URL of the JAAS authentication service. For example:

```
globals:
  # ...
  auth.url: <protocol>://<host>:<port>/FTL-JAAS/login
```

If the authentication service enables TLS, then its URL specifies the `https://` protocol, and you must supply these additional parameters to the FTL server:

```
auth.trust: <auth_service_public_cert_location>
auth.user: <ftl_server_user_name>
auth.password: <ftl_server_password>
```

JAAS Login Modules

TIBCO FTL software supports these sample login modules.

- `org.eclipse.jetty.jaas.spi.JDBCLoginModule`
- `org.eclipse.jetty.jaas.spi.PropertyFileLoginModule`
- `org.eclipse.jetty.jaas.spi.DataSourceLoginModule`

- `org.eclipse.jetty.jaas.ldap.LdapLoginModule`

For more information about login modules and how to configure them, see “JAAS Support - Eclipse” in *Jetty: The Definitive Reference*.

To implement your own login module, see “Custom Login Modules” in *Oracle Application Server Containers for J2EE Security Guide*.

Using the external custom HTTP / HTTPS based authentication service

You can customize the **http/https** based authentication service similar to JAAS based Authentication service.

- Custom authentication Service has to respond to `/auth` rest request, and return the roles associated with the user.
- See samples `/src//golang/advanced/src/tibco.com/ftl-sample/`
 - `tibffauthsvc`
 - `security`

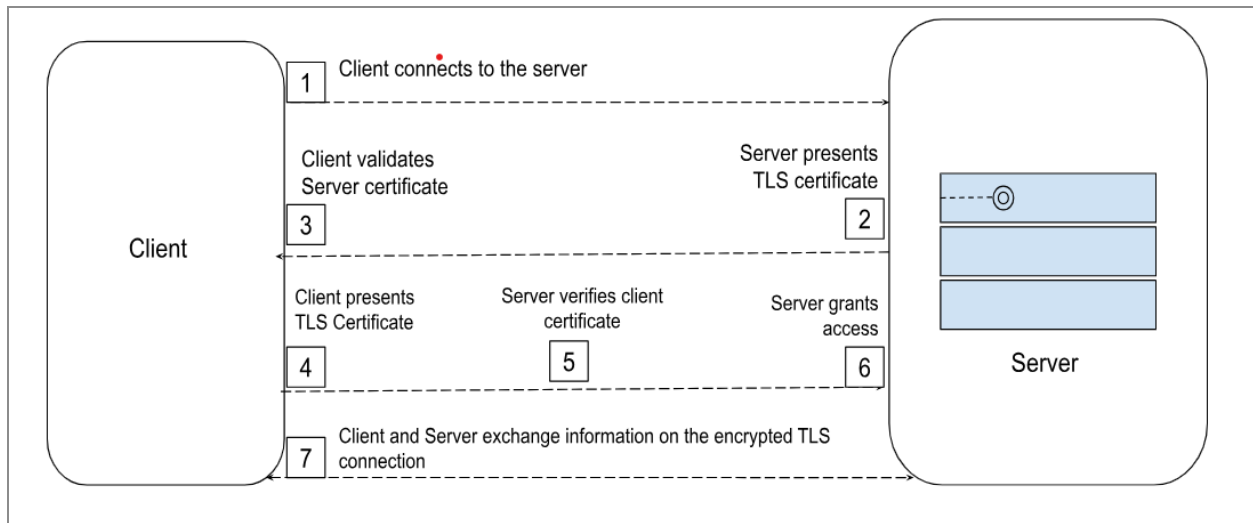
Using the Built in mTLS Based Authentication Service

mTLS (Mutual transport layer Security) allows client-to-server or server-to-server connections to authenticate each other during the TLS handshake. mTLS in FTL requires user-defined certificates.

For details, see: [Enabling TLS for FTL Server](#)

The picture below shows interactions between client and server during mTLS based authentication.

Figure 4: Built in mTLS based authentication



In order to enable mTLS for incoming connections, the 'auth.providers' in the FTLServer yaml configuration must include '**mtls**' as one of the providers. In addition, configure the "tls.server.trust.file parameter in the FTL server yaml configuration file.

For details, see [FTL Server Configuration Parameters](#).

Ensure that TLS with user-defined certificates is also configured, see [Enabling TLS for FTL Server](#)

When a client or another FTL server connects to FTL server and sends its client certificate, FTL server will verify the client certificate using `tls.server.trust.file`.

If the certificate is valid, FTL server will then parse the common name (CN) of the certificate. The CN must be a string in the following format:

```
<username>:<role1>[,<role2>,...]
```

For example, here are some valid common names:

```
admin:ftl-admin
internal:ftl-internal
subscribe-user:ftl,sub
```

The username and roles in the CN string become the username and roles for the incoming client or FTL server.

See `samples/yaml/mtls` in the FTL installation for example yaml configurations and certificates. See `samples/yaml/certs` in the FTL installation for an example of how to generate certificates for mTLS.

Some key points to consider while using mTLS

- When accessing the REST API, the REST client may authenticate itself with a client certificate
- mTLS is not supported for the UI, so another authentication provider must be configured to support UI users.
- mTLS is not supported for eFTL clients or the eFTL REST API.

Using the built in OAuth 2.0 based authentication service

FTL supports authenticating clients using OAuth 2.0 access tokens. When connecting to an FTL server configured with OAuth 2.0 authentication, an FTL client must authenticate itself to the server by presenting an access token issued by an OAuth 2.0 authorization server.

This access token must be a signed JSON Web Token (JWT) that includes claims that the FTLServer can extract to validate that the client has the right FTL role. See <https://datatracker.ietf.org/doc/html/rfc7519> for more information on what constitutes a signed JWT token.

The FTL server validates the access token's signature and claims and accepts or rejects the connection request accordingly. If this authentication process is successful, the FTL client is allowed to connect and access grants based on the FTL role specified in the access token.

In order for an OAuth 2.0 authorization server to issue an access token with the expected claims, the relevant FTL user and group information must be made available to it. Depending on your OAuth 2.0 provider, there may be a number of options available for achieving this. For example, you may be able to define FTL users and groups directly in your provider, or you may be able to integrate your provider with an external authentication service such as LDAP. Refer to your OAuth 2.0 provider's documentation for instruction.

Obtaining an Access Token

FTL and eFTL clients require an access token to connect to an FTL server configured with OAuth 2.0 authentication. Additionally, an FTL server itself may require an access token to connect to another FTL server or a TIBCO messaging product that is configured for OAuth 2.0 authentication.

i Note: FTL only supports access tokens in the form of signed JWTs with asymmetric validation keys. Unsigned JWTs, or JWTs with symmetric validation keys are not supported.

The FTL client APIs, eFTL Client APIs and the FTL server can be configured to use either the Client Credentials grant or Resource Owner Password Credentials grant for obtaining OAuth 2.0 access tokens.

FTL and eFTL clients can set up client side properties that then allow FTL/eFTL client library to retrieve OAuth 2.0 access tokens from the specified OAuth 2.0 provider. Clients may also provide the access tokens that are externally obtained.. These OAuth 2.0 access tokens are then validated by the FTLServer.

Client Credentials Grant

In the Client Credentials grant, the FTL or eFTL client (or FTL server) presents a client ID and client secret to the OAuth 2.0 authorization server. The authorization server uses these credentials to authenticate the FTL client or eFTL client (or FTL server) and issues it an access token.

This is the preferred grant type for both the FTL client and FTL server.

Resource Owner Password Credentials Grant

In the resource owner password credentials grant, the FTL client (or eFTL Client, or EMS server) first authenticates itself to the OAuth 2.0 authorization server by presenting a client ID and client secret, then provides an FTL user and password for validation by the authorization server. If both authentication operations are successful, the authorization server issues an access token to the client.

Refresh tokens are supported with this grant type. If the authorization server issues a refresh token along with the requested access token, the FTL client (or eFTL Client or FTL

server) uses the refresh token instead of the grant for requesting the next access token. If it fails to obtain a new access token using the refresh token, it try's again using the grant.

This grant type has been deemed legacy and is expected to be removed in a future update to the OAuth 2.0 framework. It should only be used in situations where the client credentials grant type is not feasible.

Using Externally-Obtained Access Tokens

In addition to the above grant types, the FTL and eFTL client APIs and the FTL server can be configured to use access tokens obtained via external means. If access tokens are directly configured in this manner, the FTL server and client APIs does not attempt to obtain access tokens using the OAuth 2.0 grants.

i Note: The FTL client APIs additionally support user-defined callbacks for obtaining access tokens. This method of obtaining access tokens requires the use of new APIs. Existing FTL client applications need to be modified to use the new APIs in order to make use of this method. Refer to the corresponding client API documentation for details.

User-defined callbacks are only available in the client APIs. This method of obtaining access tokens is not supported in the FTL server.

Access Token Expiration

OAuth 2.0 JWT access tokens can have an expiration time specified through the 'exp' claim. The FTL server enforces access token expiration by disconnecting the associated FTL client (or eFTL client or FTL server).

i Note: FTL server cannot enforce access token expirations in the following scenarios:

- Authentication is enabled, but TLS is not enabled, and the FTL-generated keystore is present.
- TLS is enabled with FTL-generated certificates.

In both scenarios, the access token is verified when the client first connects, but FTL server does not disconnect the client when the token expires. It is possible to migrate away from

both of the above configurations. See [Eliminating the FTL Generated Keystore](#) or [Eliminating FTL generated certificates](#). Once the relevant procedure is complete, access token expirations can be enforced by FTL server.

FTLServer OAuth 2.0 Configuration

To enable OAuth 2.0 authentication, the `auth.providers` parameter in the FTL server yaml configuration file must contain `oauth2` as one of the providers.

In addition, set the following parameters in the FTL server yaml configuration file:

For details, see [FTL Server Configuration Parameters](#)

For an example, see `samples/yaml/oauth2` in the FTL installation directory.

Single Sign-On with OAuth 2.0

If OAuth 2.0 authentication is enabled, FTL server can redirect the administrative UI to the user's oauth server for authentication. The browser will take the user through the oauth authorization code flow.

To enable single sign-on, set the following parameters in the FTL server yaml configuration file:

- `oauth2.ui.endpoint.auth`
- `oauth2.ui.endpoint.token`
- `oauth2.ui.client.id`
- `oauth2.ui.client.secret`
- `oauth2.ui.endpoint.logout`

For details, see [FTL Server Configuration Parameters](#)

Authenticating to FTL Server

FTL server can be configured with one or more authentication providers, allowing FTL server to understand basic authentication, mTLS authentication, oauth2 authentication, or all three. When accepting incoming connections, FTL server can accept different

authentication modes for different connections, as long as an appropriate auth provider is configured.

In addition, each client and FTL server must be configured to authenticate itself to FTL servers. When making a connection to an FTL server, the client or server must choose exactly one of the possible authentication modes (basic, mTLS, or oauth2). For FTL server, this configuration (for outgoing connections) is independent of the auth provider configuration (for incoming connections). For more information, see [FTL Server Configuration Parameters](#)

Authenticating with Basic Authentication

FTL clients: Pass the username and password as properties to the realm connect call.

For example, in C API, pass `TIB_REALM_PROPERTY_STRING_USERNAME` and `TIB_REALM_PROPERTY_STRING_USERPASSWORD` to `tibRealm_Connect`

eFTL clients can pass the username and password as properties to the connect call.

For example, in C API, set the `username` and `password` options when calling `tibftl_Connect`.

FTL servers: in the yaml configuration file, set `user` and `password` in the `ftlserver.properties` section for each FTL server. See Authenticating to other FTL Servers in [FTL Server Configuration Parameters](#)

Administrative tools:

- For the REST API, include an “Authorization” header in HTTP requests. `<credentials>` is the base64 encoding of the string “`<username>:<password>`”.

```
Authorization: Basic <credentials>
```

- If using `tibftladmin`, specify the “`--user`” and “`--password`” command line parameters. See [FTL Administration Utility](#).

Authenticating with mTLS

FTL clients: pass the client cert, private key, and private key password as properties to the realm connect call.

For example, in C API, pass `TIB_REALM_PROPERTY_STRING_CLIENT_CERT`, `TIB_REALM_PROPERTY_STRING_CLIENT_PRIVATE_KEY`, and `TIB_REALM_PROPERTY_STRING_CLIENT_PRIVATE_KEY_PASSWORD` to `tibRealm_Connect`

eFTL clients: mTLS is not supported.

FTL servers: in the yaml configuration file, set `tls.client.cert`, `tls.client.private.key`, and `tls.client.private.key.password` in the `ftlserver.properties` section for each FTL server. See Authenticating to other FTL Servers in [FTL Server Configuration Parameters](#)

Administrative tools:

- mTLS is not supported for the UI or the eFTL REST API.
- For the FTL REST API, configure the TLS provider to present a client certificate when connecting to FTL server.
- If using `tibftladmin`, specify the `--tls.client.cert`, `--tls.client.private.key`, and `--tls.client.private.key.password` command line parameters. See [FTL Administration Utility](#)

Authenticating with OAuth 2.0

Following are the four options for FTL clients

- Client credentials grant: pass the oauth2 token endpoint, client id, client secret, and (if needed) trust file to the realm connect call. The FTL library will fetch and refresh the token as needed. For example, in C API, pass the following to `tibRealm_Connect`
 - `TIB_REALM_PROPERTY_STRING_OAUTH2_SERVER_URL`,
 - `TIB_REALM_PROPERTY_STRING_OAUTH2_CLIENT_ID`
 - `TIB_REALM_PROPERTY_STRING_OAUTH2_CLIENT_SECRET`,
 - `TIB_REALM_PROPERTY_STRING_OAUTH2_SERVER_TRUST_FILE`, if needed.
- Password credentials grant: in addition to the parameters for client credentials grant, pass a username and password to the realm connect call. The FTL library will fetch and refresh the token as needed. For example, in C API, in addition to the above parameters, pass `TIB_REALM_PROPERTY_STRING_USERNAME` and `TIB_REALM_PROPERTY_STRING_USERPASSWORD` to `tibRealm_Connect`.
- Long-lived access token: pass an oauth2 access token (a signed JWT) directly to the

realm connect call. The token must be valid for the lifetime of the application. For example, in C API, pass `TIB_REALM_PROPERTY_STRING_OAUTH2_ACCESS_TOKEN` to `tibRealm_Connect`

- Access token callback: set a callback in the properties object. The callback will be invoked by the FTL library whenever a new token is needed. For example, in C API, call `tibProperties_SetOAuth2TokenFetchCallback` and pass the properties object to `tibRealm_Connect`.

eFTL clients: there are four choices. Authentication with oauth2 is not supported for the javascript or python API.

- Client credentials grant: pass the oauth2 token endpoint, client id, client secret, and (if needed) trust file to the connect call. The eFTL library will fetch and refresh the token as needed. For example, in C API, set the `oAuth2ServerUrl`, `oAuth2ClientId`, `oAuth2ClientSecret`, and `oAuth2TrustStore` options when calling `tibftl_Connect`.
- Password credentials grant: in addition to the parameters for client credentials grant, pass a username and password to the connect call. The eFTL library will fetch and refresh the token as needed. For example, in C API, in addition to the above parameters, set “username” and “password” when calling “`tibftl_Connect`”.
- Long-lived access token: pass an oauth2 access token (a signed JWT) directly to the connect call. The token must be valid for the lifetime of the application. For example, in C API, set the `oAuth2AccessToken` option when calling `tibftl_Connect`.
- Access token callback: pass a callback to the connect call. The callback will be invoked by the eFTL library whenever a new token is needed. For example, in C API, set the “`oAuth2TokenFetchCallback` and `oAuth2TokenFetchCallbackArg` options when calling `tibftl_Connect`.

FTL servers: there are three choices. See “Authenticating to other FTL Servers” in [FTL Server Configuration Parameters](#)

- Client credentials grant: in the yaml configuration file, set `oauth2.svr.endpoint.token`, `oauth2.svr.client.id`, `oauth2.svr.client.secret` and (if needed) `oauth2.provider.trust.file` in the “`ftlserver.properties`” section for each FTL server.
- Password credentials grant: in addition to the parameters for client credentials grant, set “user” and “password” in the “`ftlserver.properties`” section for each FTL server.
- Long-lived access token: in the yaml configuration file, set “`oauth2.access.token`” (a

signed JWT) in the “ftlserver.properties” section for each FTL server. The token must be valid for the lifetime of the FTL server.

Administrative tools:

- Single sign-on may be configured for the UI. See “Single Sign-On with OAuth2” in [FTL Server Configuration Parameters](#)
- For the REST API, include an “Authorization” header in HTTP requests. <token> must be a signed JWT.

```
Authorization: Bearer <token>
```

- If using “tibftladmin”, specify the “--oauth2.token” command line parameter (a signed JWT). See [FTL Administration Utility](#).

Authorization

FTL Server Authorization Groups

A username may belong to several authorization groups (also known as *roles*). The following table specifies authorization group requirements.

Authorization Groups

Authorization Group	Usage
ftl	FTL servers require client programs to authenticate with usernames in the authorization group ftl.
ftl-admin	Authenticated users in the authorization group ftl-admin can execute administrative operations, modify the realm definition, and view monitoring pages.
ftl-internal	FTL servers require affiliated FTL servers to authenticate with a username in the authorization group ftl-internal.

i Note: As of Release 6.0, the authorization groups `ftl-primary`, `ftl-satellite`, `ftl-backup`, `ftl-dr` are obsolete. For each of these, use `ftl-internal` instead.

Mapping Authorization Groups

FTL allows administrators to map their own roles to FTL roles. This can be done via the `auth.rolemap` file that can contain a mapping specific to the authentication provider.

Rules on role mapping file

- Role mapping within 'ldap' section applies to ldap or ldaps
e.g in the example below
 - [ldap]:
 - FTL-Admin: ftl-admin
- Only one mapping per authentication provider is allowed, for example you cannot have two separate authentication sections for ldap/ldaps or mtls or oauth2

Note: Syntax Summary for the mapping file

- A role mapping will have sub sections per authentication type
- Each subsection starts with a square bracket `[`, **followed by the authentication provider name and a closing square bracket `]`.**
 - Allowed authentication provider names
 - ldap or ldaps
 - mtls
 - oauth2
- A line with a header with this syntax **[<authentication provider]** defines the subsection of the authentication type and subsequent lines until the next **[<authentication provider]** section would be role mapping associated with that authentication provider.
- There can be only one subsection per authentication provider.
- A role mapping within the authentication provider section defines the mapping from

the user defined role to FTL built in role that's pertinent to the authentication provide

- Delimit the user-defined role with a required colon. (e.g FTL-Admin: ftl-admin)
- You may add optional space characters after the colon. The FTL built-in role begins with the first non-whitespace character after the colon.
- Delimit the FTL role with a comma to add additional FTL built -in roles that map to the same user defined role to FTL roles.
- Separate authorization roles or groups with a comma only (spaces are not valid).

Here is an example of role mapping file named rolemap.txt

```
[ldap]
FTL-Admin: ftl-admin
[mtls]
group1: ftl-admin, ftl-internal
group2: ftl
[oauth2]
oauth2-admin: ftl-admin
oauth2-apps: ftl
```

Permissions

Administrators can secure the messaging infrastructure by setting permissions at the cluster and store level to grant access to objects for a given user/role.

Administrators enable permissions by setting the realm property `enable_permissions` to `true`.

The requirements to use permissions follow:

- All persistence transports must be marked secure.
- Clients must be using FTL 6.8.0 or later.
- Authentication must be enabled (via the `auth.providers` parameter in the `ftlserver` configuration file).
- Once authentication is enabled:

- Assign permissions via the Users Grid and Roles Grid. See [Configuring Permissions](#).
- On the Realm Properties Details Panel, **Permissions** is set as **Disabled** by default and must be set as **Enabled**. See [Realm Properties Details Panel](#).
- When `enable_permissions` is true, endpoint store inboxes are always enabled. See [Endpoint Store Inboxes](#).

For details on the preferred method to upgrade to FTL 6.8.0 , see [Migrating to FTL 6.8.0 when Using Permissions](#) .

Enable Permissions

To enable permissions, set the `enable_permissions` flag in the realm properties to true.

When the `enable_permissions` flag is set to true, permissions are enforced at every persistence cluster defined in the realm. By default, a user has no permissions; permissions must be explicitly granted to users or roles as described in the next section, [Assign Permissions](#). There is one exception: users with the predefined `ftl-internal` role always have all permissions. For this reason the `ftl-internal` role should not be assigned to ordinary users.

Endpoint store inboxes are enabled when either of the following are true. (See [Endpoint Store Inboxes](#).)

- `use_endpoint_store_for_inbox` is true
- `enable_permissions` is true

Setting `enable_permissions` to true does not change the value of `use_endpoint_store_for_inbox`, but endpoint store inboxes are enabled. If `enable_permissions` is set back to false, then endpoint store inbox behavior depends on the existing value of `use_endpoint_store_for_inbox`. The value is not changed.

When using permissions with the request-reply API, the requestor has an implicit inbox subscriber on the reply endpoint's store. (By default the reply endpoint is the same as the publisher's endpoint.) Because endpoint store inboxes are automatically enabled, this subscription is created in the reply endpoint's store. Therefore:

- Requests (using `tibPublisher_Request`) require publish permissions on the endpoint's store and subscribe permissions on the reply endpoint's store (which could be a different endpoint)
- Replies (using `tibPublisher_Reply`) require 'publish' permissions

Assign Permissions

The administrator assigns permissions at the store or persistence cluster level.

Each store and persistence cluster in the realm configuration has a permission map (the `acl` field) that associates permissions with usernames and/or roles.

In the permissions map, a username or role can be associated with a list of string values representing permissions. Any FTL client with the specified username or role will have the associated permissions when interacting with the given persistence store or persistence cluster. If an FTL client matches multiple entries in the permissions map (a username and/or any number of roles), the FTL client has the union of all specified permissions.

Persistence cluster and persistence store permissions do not overlap. Persistence stores do not inherit permissions defined at the persistence cluster level.

The following permissions may be defined at a persistence cluster:

- lock

The following permissions may be defined at a persistence store:

- publish
- subscribe
- map

See [Required Permissions for API Calls](#).

Adding permissions to a username or role does not require a restart of any FTL components.

Revoke Permissions

Permissions associated with a username or role at any persistence cluster or persistence store may be removed at any time. Removing permissions does not require a restart of any FTL components.

Permission revocation is not enforced synchronously. At some later time FTL will enforce the new restrictions.

See [Required Permissions for API Calls](#).

Even when the new restrictions are applied, a subscriber whose subscribe permission was revoked may receive up to prefetch number of messages.

Monitoring Gateway Service

TIBCO® Messaging Monitor for TIBCO FTL® runs as an ordinary FTL client. The subscribe permission needs to be granted to the appropriate user and/or role on the built-in monitoring and logging stores (`ftl.system.mon.store` and `ftl.system.log.store`). The same is true of any application that subscribes on the monitoring or logging endpoints. It is not possible to grant the publish or map permissions on the monitoring and logging stores.

Configuring Permissions

Permissions for users and roles can be configured from the user interface on the Users grid and Roles grid.

In edit mode, you can add and delete users and roles.

For information on enabling, assigning, and revoking permissions as well as monitoring permissions, see [Authorization](#).

For the consequences of modifying the permission configurations, see [Required Permissions for API Calls](#).

Users Grid

Column	Description
User	The FTL client's username.
Resource	The identifier of the defined resource. Example: Resource = <code>ftl.nonpersistent.store</code>
Resource Type	The type of resource including persistence stores and eFTL channels. Example: Resource Type = Persistence Store
Resource Belongs To	The name of the object. The resource belongs to a persistence cluster or eFTL cluster, so in that sense it belongs to a service that is servicing this cluster. Example: Resource Belongs To = <code>ftl.default.cluster</code>

Column	Description
Permissions	<p>The permissions granted.</p> <p>Requirements:</p> <ul style="list-style-type: none"> • The FTL server must have TLS and authentication enabled. See Authorization. • The Realm Properties Details Panel, Permissions must be set as Enabled. See Realm Properties Details Panel. <p>At the persistence cluster level, the value is <code>lock</code>.</p> <p>At the persistence store level, the value can be:</p> <ul style="list-style-type: none"> • <code>publish</code> • <code>subscribe</code> • <code>map</code>

Roles Grid

Column	Description
Roles	The FTL client's role.
Resource	<p>The identifier of the defined resource.</p> <p>Example: Resource = <code>ftl.nonpersistent.store</code></p>
Resource Type	<p>The type of resource including persistence stores and eFTL channels.</p> <p>Example: Resource Type = Persistence Store</p>
Resource Belongs To	<p>The name of the object.</p> <p>Example: Resource Belongs To = <code>ftl.default.cluster</code></p>
Permissions	<p>The permissions granted.</p> <p>At the persistence cluster level, the value is <code>lock</code>.</p>

Column	Description
	At the persistence store level, the value can be: <ul style="list-style-type: none"> • publish • subscribe • map

Required Permissions for API Calls

Permissions are granted to the FTL client's user name and/or roles. To set user and role permissions from the FTL user interface, see [Configuring Permissions](#).

When an application interacts with a persistence store or cluster, FTL determines if that interaction is allowed, based on the permissions granted to the FTL client's username and/or roles. Mostly commonly this interaction results from an API call, but all interactions are regulated regardless of the source.


Interactions involving locks are regulated by permissions set at the persistence cluster level.

Interactions involving publishers, subscribers, maps, and stored data are regulated by permissions set at the persistence store level.

FTL server logs all authorization failures at the loglevel `ac1:verbose`.

Where possible, if an FTL client is not authorized to take a certain action, the API call will fail immediately regardless of any retry duration.

The following table shows the required permissions for various API calls.

 **Note:** Only the C APIs are listed, but the same holds true for all client APIs.

Permissions for API Calls

Operation	FTL API	FTL Permission
acquire lock	Implicit via map calls <code>withLock</code>	lock

Operation	FTL API	FTL Permission
return lock	tibLock_Return tibLock_Destroy	lock
close map	tibMap_Close	map
create map	tibRealm_CreateMap	map
delete map	tibRealm_RemoveMap	map
map get	tibMap_Get tibMap_GetMultiple	map
map get size	tibMap_GetSize	map
map iterate	tibMap_CreateIterator tibMapIterator_Next	map
map remove	tibMap_Remove	map
map remove all	tibMap_RemoveAll	map
map set	tibMap_Set tibMap_SetMultiple	map
close publisher	tibPublisher_Close	publish
create publisher	tibPublisher_Create	publish
publish	tibPublisher_Send tibPublisher_SendToInbox tibPublisher_SendMessages	publish
send reply	tibPublisher_SendReply	publish

Operation	FTL API	FTL Permission
send request	tibPublisher_SendRequest Note: Sending the request requires the publish permission on the endpoint's store. Receiving the reply requires the subscribe permission on the reply endpoint's store (which could be a different store)	publish AND subscribe
acknowledge	tibEventQueue_Dispatch (auto) tibMessage_Acknowledge (explicit) tibSubscriber_AcknowledgeMessages (explicit)	subscribe
close subscriber	tibSubscriber_Close	subscribe
durable create	Implicit via tibSubscriber_Create	subscribe
dynamic durable destroy	tibRealm_Unsubscribe tibRealm_UnsubscribeEx	subscribe
start subscriber	tibEventQueue_AddSubscriber	subscribe
stop subscriber	tibEventQueue_RemoveSubscriber	subscribe
subscribe	tibSubscriber_Create tibSubscriber_CreateOnInbox	subscribe
rewind	tibRealm_RewindSubscription().	subscribe
create browser	tibBrowser_Create()	subscribe
browse message	tibBrowser_Next()	subscribe

Operation	FTL API	FTL Permission
delete browsed message	<code>tibBrowser_DeleteMessage()</code>	subscribe
close browser	<code>tibBrowser_Close()</code>	subscribe

Migrating to FTL 6.8.0 when Using Permissions

If you are migrating from a version of FTL prior to 6.8.0 and want to set up permissions, complete the steps in the following order to minimize the number of application restarts.

1. Ensure that TLS and authentication are enabled.
2. Ensure that all persistence transports are secure. Changing this will require a restart of persistence services and clients.
3. Upgrade the servers to 6.8.0.
4. Upgrade the clients to 6.8.0.
5. Set `enable_permissions` to true and configure appropriate permissions for users and/or roles. This will require a restart of any publisher or subscriber using server-based inboxes (via a dedicated inbox store, `ftl.system.inbox.store` or `ftl.routing.inbox.store`) because endpoint store inboxes are now required. See [Endpoint Store Inboxes](#).
6. Verify that stores are using the permissions you have configured. If no permissions are set up, users will not have access to publish/subscribe to a store. The loglevel `acl:verbose` may be set at the FTL server (specifically, the persistence service) to identify authorization failures.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join [TIBCO Community](#).

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [Product Documentation website](#), mainly in HTML and PDF formats.

The [Product Documentation website](#) is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO FTL® - Enterprise Edition is available on the [TIBCO FTL® - Enterprise Edition Product Documentation](#) page.

TIBCO eFTL™ Documentation Set

TIBCO eFTL software is documented separately. Administrators use the FTL server GUI to configure and monitor the eFTL service. For information about these GUI pages, see the documentation set for TIBCO eFTL software.

How to Contact Support for TIBCO Products

You can contact the Support team in the following ways:

- To access the Support Knowledge Base and getting personalized content about products you are interested in, visit our [product Support website](#).
- To create a Support case, you must have a valid maintenance or support contract with a Cloud Software Group entity. You also need a username and password to log in to the [product Support website](#). If you do not have a username, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME CLOUD SOFTWARE GROUP, INC. (“CLOUD SG”) SOFTWARE AND CLOUD SERVICES EMBED, BUNDLE, OR OTHERWISE INCLUDE OTHER SOFTWARE, INCLUDING OTHER CLOUD SG SOFTWARE (COLLECTIVELY, “INCLUDED SOFTWARE”). USE OF INCLUDED SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED CLOUD SG SOFTWARE AND/OR CLOUD SERVICES. THE INCLUDED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER CLOUD SG SOFTWARE AND/OR CLOUD SERVICES OR FOR ANY OTHER PURPOSE.

USE OF CLOUD SG SOFTWARE AND CLOUD SERVICES IS SUBJECT TO THE TERMS AND CONDITIONS OF AN AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER AGREEMENT WHICH IS DISPLAYED WHEN ACCESSING, DOWNLOADING, OR INSTALLING THE SOFTWARE OR CLOUD SERVICES (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH LICENSE AGREEMENT OR CLICKWRAP END USER AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE SAME TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of Cloud Software Group, Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, FTL, eFTL, and Rendezvous are either registered trademarks or trademarks of Cloud Software Group, Inc. in the United States and/or other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only. You acknowledge that all rights to these third party marks are the exclusive property of their respective owners. Please refer to Cloud SG’s Third Party Trademark Notices (<https://www.cloud.com/legal>) for more information.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

Cloud SG software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the “readme” file for the availability of a specific version of Cloud SG software on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. CLOUD SG MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S), THE PROGRAM(S), AND/OR THE SERVICES DESCRIBED IN THIS DOCUMENT AT ANY TIME WITHOUT NOTICE.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "README" FILES.

This and other products of Cloud SG may be covered by registered patents. For details, please refer to the Virtual Patent Marking document located at <https://www.cloud.com/legal>.

Copyright © 2009-2024. Cloud Software Group, Inc. All Rights Reserved.