SAS® Event Stream Processing 4.3: Security

Enabling Encryption on Sockets

Overview to Enabling Encryption

You can enable encryption on TCP/IP connections within an event stream processing engine. Specifically, you can encrypt the following:

- connections that are created by a client using the C, Java, or Python publish/subscribe API to connect to an event stream processing server.
- connections that are created by an adapter connecting to an event stream processing server.
- connections that are created by a file and socket connector or adapter that acts as a socket client or server. In this case, the TCP peer can be another file and socket connector or adapter, or a third-party socket application.

  Note: Suppose a file and socket connector or adapter connects to a SAS LASR Analytic Server. Suppose further that the server runs on a Hadoop Distributed File System (HDFS) name node in order to access SAS HDAT files. In this case, encryption is not supported.

You must meet the following requirements in order to enable encryption:

- The OpenSSL libraries must be installed on all computer systems that run the client and server. When you install SAS Event Stream Processing Encryption and Authentication Overlay, you install OpenSSL.
- The `DFESP_SSLPATH` environment variable must be defined with the path to the OpenSSL shared object or DLL.

  Note: For a Java publish/subscribe client or adapter, the value of `DFESP_SSLPATH` is irrelevant. SSL is supported natively in Java. Nevertheless, you must define a value of `DFESP_SSLPATH`.

- The proper SSL certificates must be installed on the client and server. If encryption is not enabled through DFESP_SSLPATH, the installation runs successfully without OpenSSL being installed.

Configuration Directory

The default location for configuration files is **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default** on Linux and **\SAS\Viya\SASEventStreamProcessingEngine\default** on Windows.

On Windows, if the environment variable PUBLIC is defined, it is placed at the beginning of the configuration directory's path, before **\SAS**.

Understanding SSL Certificate Requirements

Consider the following SSL certificate requirements in order to enable encryption:

- The required SSL certificates differ for client and server. You must copy the needed certificate files into the configuration directory.
- The server is an engine with publish/subscribe enabled or a file and socket connector or adapter running as a server. The server requires the following files in the configuration directory:
  - **server.pem**

    This file must contain a concatenation of a certificate and private key.
  - **key.passphrase** (only when the key is password-protected)
- A client is a C, Java, or Python publish/subscribe client, an adapter, or a file and socket connector or adapter running as a client. The client requires the **ca.pem** file in the configuration directory. This file must contain a certificate used to verify the received server side certificate.

  For certificates signed by a Certificate Authority (CA), the certificate of the signer must be present. For self-signed certificates, the certificate can be a copy of the certificate in **server.pem**.

Production traffic must use only certificates that are signed by a CA.

Note: When you start an event stream processing server with **http-pubsub** enabled, it runs both client and server internally. Thus, if encryption is enabled, your configuration directory must contain the client and server certificates.

The event stream processing client and server forces the negotiated encryption protocol and cipher suite to be TLSv1.2 compliant.

Understanding the SSL Handshake Process

When DFESP_SSLPATH is defined but SSL certificates cannot be found or the OpenSSL library cannot be loaded, a fatal error is logged at start-up. If there are no start-up errors, the publish/subscribe server indicates whether SSL is enabled or disabled through an INFO level log message logged when the server starts up.

A client that is enabled for SSL logs a handshake-status INFO-level message when it initiates its connection. When a client or server is negotiating SSL and its peer is not, or when the SSL handshake itself fails, the connection fails and an error message is logged.

Enabling Authentication on Socket Connections

Overview

You can require authentication for TCP/IP clients that connect to an event stream processing engine. You must install the OpenSSL libraries on your event stream processing server in order to implement authentication. You can install the SAS Event Stream Processing Encryption and Authentication Overlay package to install OpenSSL.

Authentication applies to the following event stream processing engine APIs:

- the publish/subscribe API
- connections created by a client that uses the C, Java, or Python publish/subscribe API to connect to an event stream processing engine
- connections created by an adapter that connects to an event stream processing engine
- the ESP server HTTPS API
- connections created by the ESP client (`dfesp_xml_client`) to communicate with the ESP server using the HTTPS protocol

To implement authentication, an event stream processing server must be enabled for publish/subscribe operations.

When enabled on a server, authentication is a global and permanent setting, so all clients that connect to that server must be authenticated. This applies to all client operations that are supported by the publish/subscribe API. It also applies to queries and all other client/server requests that establish a unique TCP connection. When authentication fails, a client is disconnected and an error message is logged on both the client and server.

Similarly, a client that requests authentication to a server that is not enabled for authentication results in client disconnection. There is a corresponding error message.

Token Method Authentication

The token method authentication mechanism requires a signed JSON Web Token obtained from an OAuth 2.0/ OpenID Connect compliant server. This token is supplied to the publish/subscribe API or adapter by the user. The token is then passed in compact serialization form (base64url encrypted) from client to server. It is parsed and validated by the server.

SAS Event Stream Processing requires that the token be obtained from a supported OAuth server. This is currently limited to the Cloud Foundry (CF) User Account and Authentication (UAA) Server. You must request a token from the CF UAA server, and then provide that token to the client.

For authentication certification, tokens are generated by invoking a REST request to a locally installed CF UAA server through curl. The REST request invokes the implicit grant with credentials flow. For more information, see the CF UAA documentation. For an example, see CF UAA Client/Server Information.

The following resources provide more information:

- JSON web tokens: https://self-issued.info/docs/draft-ietf-oauth-json-web-token.html
- OAuth 2.0: https://tools.ietf.org/html/rfc6749
- OpenID Connect: http://openid.net/specs/openid-connect-core-1_0.html
- CF UAA: https://github.com/cloudfoundry/uaa

Authentication Using a User Name and Password

This authentication mechanism requires simple user name and password credentials to be provided by the user. These credentials must be valid when passed by the event stream processing server to a SASLogon service.

The sequence of processing is as follows:

1. A user supplies credentials to the publish/subscribe API, adapter, or HTTP client.
2. Credentials are passed unmodified to the event stream processing server.
3. The server passes credentials unmodified in a REST request to the configured SASLogon service.
4. The server returns the result of the authentication request to the client.

SAS Event Stream Processing requires that the user at a minimum provide a user name in the publish/subscribe URL. Alternatively, the password can be provided directly in the publish/subscribe URL. Otherwise, the event stream processing client API searches an **`.authinfo`** or **`.netrc`** file in the client's local filesystem for a password that matches the provided user name. Either way, the password can be clear text or SAS encoded.

Kerberos Authentication

Enabling Kerberos authentication requires that the SAS Event Stream Processing server and client reach a Kerberos server. That Kerberos server must support a service name to be used by the server and client.

When these conditions are met, a user can pass the same Kerberos service name to the SAS Event Stream Processing server and client. The appropriate Kerberos exchange is performed between client and server when the client connects. The client builds the full service principal name by combining the service name passed by the user with the host contained in the publish/subscribe URL passed by the user.

Token Method Server Requirements

Authentication is enabled on an event stream processing server by passing a client ID string when initializing the engine. In the C++ modeling API, this is a parameter in the `dfESPengine::initialize()` call. To enable authentication, replace the current `pubsub_ENABLE(`*`portNum`*`)` parameter with `pubsub_ENABLE_OAUTH(`*`portNum`*`, `*`clientId`*`)`.

When running an ESP server, enable authentication by including the **`-auth oauth://`*`clientId`*** command-line parameter.

This *clientId* must match the CF UAA client_id used when requesting a token from the CF UAA server. For more information, see Token Validation.

In addition, the server must contain the public key used to sign the token in its local file system in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/pubkey.pem`** (Linux) or **`%PUBLIC%\SAS\Viya\SASEventStreamProcessingEngine\default\pubkey.pem`** (Windows). For more information, see Token Validation.

Any error in token validation causes the server to return an error code to the client. The client then disconnects from the server.

Token Method Client Requirements

A client requests an authenticated connection by passing a token to the server. You can provide the token to the client in one of two ways:

- Pass it in the publish/subscribe or adapter URL through the following optional element:

`?oauth_token`

This element must follow the *host*`:`*port* part of the URL, as follows:

`dfESP:/`*host*`:`*port*`?oauth_token=`*token*..... The remainder of the URL is the same.

- Specify the complete path and filename of a file on the local file system that contains the token. When using the publish/subscribe API, call the corresponding C, Java, or Python publish/subscribe API method. The C method is `C_dfESPpubsubSetTokenLocation()` , and the Java and Python method is `setTokenLocation()`. When running an adapter, use the corresponding optional adapter configuration switch.
- When running an ESP client, pass the token using the **`-auth oauth-token`** or **`-auth oauth-token-url`** command-line parameter.

Using both methods simultaneously is not allowed. It generates a publish/subscribe API error. The client passes the token opaquely to the server and waits for token validation results. If successful, the connection is established and further client server operations proceed normally. If unsuccessful, the client disconnects from the server.

Token Validation

The server validates multiple items in a received token:

Validated Item	Description	
Token Signature	The server uses the OpenSSL libraries and the public key in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/pubkey.pem`** (Linux) or **`%PUBLIC%\SAS\Viya\SASEventStreamProcessingEngine\default\pubkey.pem`** (Windows) to verify the signature in a received token. This public key must be the same key as the public key in the public-private key pair that is configured on the CF UAA server. You must generate, secure, and manually copy this key to the server. For more information, see CF UAA Client/Server Information.	
Claims	`aud`	The client ID configured on the server must match the `aud` claim contained in all tokens received by the server. The value of the `aud` claim in a token is determined by the CF UAA client ID included in a request to the CF UAA server to obtain that token. A CF UAA administrator must configure an event stream processing specific client ID. You must specify that same ID when you start an authentication-enabled server. You must also specify that ID when you request a token to be used by a client connecting to that server. Server-specific privileges can be enforced by requiring server-specific client IDs for connections to that server.
	`exp`	Token validation fails when the token is expired.

Server Requirements for Authentication Using a User Name and Password

This method of authentication is enabled on an event stream processing server by passing a SASLogon services URL when initializing the engine. In the C++ modeling API, this is through a parameter in the `dfESPengine::initialize()` call.

To enable authentication, replace the current `pubsub_ENABLE(portNum)` parameter with `pubsub_ENABLE_SASLOGONOAUTH(portNum, sasLogonURL)`. When running an ESP server, enable this method of authentication by including the **-auth saslogon://*sasLogonURL*** command–line parameter.

In order for the event stream processing server to load and use the required SAS encryption libraries, you must set environment variable `TKERSA2_LIB_PATH` to the location of your SASFoundation libraries. Any error in authenticating the user credentials causes the server to return an error code to the client. The client then disconnects from the server.

Client Requirements for Authentication Using a User Name and Password

A client requests an authenticated connection by passing a user name and password to the server. You can provide these to the client in one of three ways:

- Pass them in the publish/subscribe or adapter URL through the following optional elements: `?username` and `?password` These elements must follow the ***host*:*port*** part of the URL, as follows: **dfESP:/ *host*:*port*?username=*username*?password=*password***. The remainder of the URL is the same.
- Pass the user name as described in the previous way, but have the API extract the matching password from a local **.authinfo** or **.netrc** file. In this case environment variable `AUTHINFO` or `NETRC` should be set to the full path and filename. If an environment variable is not set, the client API looks for the file in the user's home directory.
- When running an ESP client, pass the user name using the **-auth saslogon://*user*** command-line parameter.

The password can be encoded, but it does not have to be. If required, encode your password using the PWENCODE procedure. Be sure to include the `{SAS00x}` prefix from the PWENCODE result in the password string passed to the event stream processing client or included in your **.authinfo** or **.netrc** file. The client passes the credentials opaquely to the server and waits for SASLogon authentication results. If successful, the connection is established and further client server operations proceed normally. If unsuccessful, the client disconnects from the server.

Server Requirements for Kerberos Authentication

Enabling Kerberos authentication on an event stream processing server requires passing a Kerberos service name when you initialize the engine. In the C++ modeling API, you initialize the engine through a parameter in the `dfESPengine::initialize()` call. To enable authentication, replace the current `pubsub_ENABLE(portNum)` parameter with `pubsub_ENABLE_KERBEROS(portNum, serviceName)`.

When running an event stream processing server, enable Kerberos authentication by including the **-auth kerberos://*serviceName*** command–line parameter. The Kerberos server environment must contain a readable KeyTab file, which is usually found in **/etc/krb5.keytab**.

Note: The event stream processing server does not support Kerberos authentication on the Microsoft Windows platform.

Client Requirements for Kerberos Authentication

A client requests an authenticated connection by passing a service name to the server. You provide this to the client in the publish/subscribe or adapter URL through the optional **?kerberos_servicename** element. This element must follow the ***host*:*port*** part of the URL. For example:

dfESP:/ *host*:*port*?kerberos_servicename=*servicename*.

The remainder of the URL is the same.

When you run an event stream processing client, pass the service name using the **-auth kerberos://*serviceName*** command-line parameter.

The server verifies that it was configured with the same service name before initiating the Kerberos protocol exchange.

You might have to run kinit before you run the client to refresh the local token cache.

If the client is a Java client, a jaas.conf file must be present in the SAS Event Stream Processing configuration directory. It should be a copy of the jaas.conf file that is already used by other Java clients on the client machine. The file must contain these lines:

```
com.sun.security.jgss.krb5.initiate {

com.sun.security.auth.module.Krb5LoginModule required
   useTicketCache=true
   renewTGT=true
   doNotPrompt=true;
};
```

If a Java client fails authentication, you can use these debug properties on the Java command line:

–Dsun.security.krb5.debug=true -Dsun.security.jgss.debug=true

If debugging results show the message "`unsupported key type found the default TGT: 18`", you might have to install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files.

CF UAA Client/Server Information

The CF UAA server is an open-source package available from GitHub. After you install it, the following additional administrative steps are highly recommended:

- Generate a new private key using OpenSSL (**openssl genrsa -out privkey.pem 1024**, for example), and then generate a public key based on that private key (**openssl rsa -pubout -in privkey.pem -out pubkey.pem**, for example). Keep these keys secure.
- Configure the CF UAA token verification-key with the public key, and the CF UAA token signing-key with the private key. Then copy the public key to all event stream processing servers that should authenticate clients using tokens that are generated by this CF UAA server.
- Configure one or more CF UAA client IDs restricted for use only by users running an event stream processing server or client
- Register CF UAA user name and password credentials for users requiring tokens for use by an event stream processing client.

Once configured, the steps required to obtain and use tokens for authenticated connections are as follows:

- To connect a client to a specific server, obtain a token from a CF UAA server configured with the same public key used by the event stream processing server. The token request must contain the same CF UAA client ID that you used to enable authentication on the server. You can choose the method of requesting a token from CF UAA.
- Extract the token from the response and provide it to your client as described in Token Method Client Requirements.
- You can reuse a single token indefinitely when connecting to the same server, as long as that token remains unexpired.

Here is an example of a REST request invoked through curl to obtain a token from a CF UAA server through an implicit grant with credentials:

curl -v -H "Accept: application/json" -H "Content-Type: application/x-www-form-urlencoded" "http://*myhost*:8080/uaa/oauth/authorize?

client_id=*myclientid*&response_type=token&scope=openid&redirect_uri=http://localhost/hello" -d "credentials=%7B%22username%22%3A%*22myusername*%22%2C%22password%22%3A%22*mypassword*%22%7D"

When the REST response contains a successful “302 Found” response, you can find the token in the `&access_token` portion of the `Location` field of the REST response.

Using Access Control

You can set up the event stream processing server to use explicit read/write permissions on engine, project, query, and window objects based on the user. You define these permissions in the file permissions.yml, which is located in your configuration directory . This file defines all permissions on all objects for all users for the server installation defined by **$DFESP_HOME**. The default installation contains a permissions.yml.sample file in your configuration directory.

Access control applies only to networked event stream processing clients, such as publish/subscribe clients, adapters, XML clients, and Streamviewer. The user who runs the server is not limited in any way.

Note: You cannot use access control on the 64–bit ARM platform.

To enable access control, the server administrator can modify permissions.yml.sample to reflect the objects in the model and the users who need access. Then, the administrator can rename the sample file to permissions.yml.

If **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/permissions.yml** (Linux) or **%PUBLIC%\SAS\Viya\SASEventStreamProcessingEngine\default\permissions.yml** (Windows) does not exist or cannot be read, then the event stream processing server runs without access control. At start-up, the event stream processing server sends an informational message that logs whether access control defined in permissions.yml is in effect.

If an event stream processing client does not provide a user name, then it is assumed to use “anonymous”. To provide a user name when connecting, an event stream processing client must authenticate against an authentication-enabled event stream processing server. Furthermore, the server must be configured for SASLogon style authentication. That is, a C++ server must use `pubsub_ENABLE_SASLOGONOAUTH(portNum, sasLogonURL)`, and an ESP server must specify **–auth saslogon** on the command line.

When a client is denied access, the server informs the client, closes the connection, and logs a warning message. The client logs an error message with the error code returned from the server.

A user whose name is not listed in permissions.yml or who attempts to access an object not listed in permissions.yml is denied access. The only supported permissions are Read and Write, and every user-object combination must have its Read and Write access explicitly defined as true or false.

In addition to users, permissions.yml can also contain groups, where group permissions are defined using the same syntax for users. Then, any user can be added to a group. A user can have explicitly defined permissions or can belong to a group but not both.

When an object is dynamically added to a running server (when an XML client loads a project, for example), the server grants Read and Write privileges to the user who created the object. The permissions.yml file is not modified.

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. Copyright © 2017, SAS Institute Inc., Cary, NC, USA. All Rights Reserved. May 2017 4.3-P1:espsec

THE POWER TO KNOW®