Understanding the eStreamer Protocol

The Secure Firewall System Event Streamer (eStreamer) uses a message-oriented protocol to stream events and host profile information to your client application. Your client can request fully-qualified events from a Management Center. Only connection events, intrusion events, intrusion event packets, and file events are available as fully-qualified events.

Your client application initiates the data stream by submitting request messages, which specify the data to be sent, and then controls the message flow from the Management Center or managed device after streaming begins.

Throughout this document, the eStreamer service on the Management Center or a managed device may be referred to as the eStreamer server or eStreamer.

Connection Specifications

- Communicates using TCP over an SSL connection (the client application must support SSL-based authentication).
- Accepts connection requests on port 8302.
- Waits for the client to initiate all communication sessions.
- Writes all message fields in network byte order (big endian).

Establishing an Authenticated Connection

Before a client can request data from eStreamer, the client must initiate an SSL-enabled TCP connection with the eStreamer service. The client can request on any configured management interface on the Management Center or managed device. Client connections do not enforce traffic channel configuration for management interfaces so that configuration can be ignored when choosing an interface for your connection. When the client initiates the connection, the eStreamer server responds, initiating an SSL handshake with the client. As part of the SSL handshake, the eStreamer server requests the client's authentication certificate, and verifies that the certificate is valid (signed by the Internal Certifying Authority [Internal CA] on the eStreamer server).

Cisco recommends that you also require your client to verify that the certificate presented by the eStreamer server has been signed by a trusted Certifying Authority. This is the Internal CA certificate included in the PKCS#12 file that Cisco provides when you register a new eStreamer client with the Management Center or managed device.

After the SSL session is established, the eStreamer server performs an additional post-connection verification of the certificate. This includes verifying that the client connection originates from the host specified in the certificate and that the subject name of the certificate contains the appropriate value. If either post-connection check fails, the eStreamer server closes the connection. If necessary, you can configure the eStreamer service so that it does not perform a client host name check.

While the client is not required to perform post-connection verification, Cisco recommends that the client perform this verification step. The authentication certificate contains the following field values in the subject name of the certificate:

Table 1: Certificate Subject Name Fields

Field	Value
title	eStreamer
generationQualifier	server

After the post-connection verification is finished, the eStreamer server awaits a data request from the client.

Establishing a Session

The client establishes a session by sending an initial Event Stream request to the eStreamer service.

For fully-qualified events you must submit the data requests in a follow-on message. This initial Event Stream request message itself is a prerequisite for all eStreamer requests.

Note The eStreamer client can request on any configured management interface on the Management Center or managed device. Client connections do not enforce traffic channel configuration for management interfaces so that configuration can be ignored when choosing an interface for your connection.

Error Message Format

Both the client application and the eStreamer service use error messages. Error messages have a message type of `1` and contain a header, an error code, an error text length, and the actual error text. Error text can contain between 0 and 65,535 bytes.

When you create custom error messages for your client application, Cisco recommends using `-1` as the error code.

The following table describes the basic error message format. The Header Version, Message Type, and Message Length are not specific to the error message type.

Bytes	Name	Data Type	Description
0-15	Header Version	uint16	Always `1`.
16-31	Message Type	uint16	Always `1`.
32-63	Message Length	uint32	Length of the error message, in bytes.
64-95	Error Code	uint32	A number representing the error.
96-111	Error Text Length	uint16	The number of bytes included in the error text field.
112 on	Error Message	Variable	The error message. Up to 65,535 bytes.

Requesting Fully-qualified Events

Instead of receiving events in the complicated binary format, we recommend that your client uses this option to request fully-qualified events in a text format such as JSON or CSV. When using this option, the majority of this document describing the binary format is irrelevant. In the SDK package, the python_client subdirectory provides sample code for using this option.

This option currently only supports requesting information for a few event types: connection events, intrusion events, intrusion packets, and file events. If you need to receive other event types in binary format, then separate client connections must be used for fully-qualified and binary event formats.

To request fully-qualified events, use the documented "Event Stream Request Message", and append a JSON-format configuration block at the end of the message. The request will include the usual five binary integers shown below, followed by the JSON-format configuration details, like:

```
<Header Version (1)> <Message Type (2)> <Message Length> <Initial Timestamp> <Request Flags>
<JSON-format Configuration Block>
```

The binary Message Length field must include the length of the binary header, plus the length of the JSON block. A terminating null character is optional after the JSON block, but if the null is included then the Message Length must account for the null character. For the Request Flags field, only bit 23 (extended event headers) is supported; all other bits should be zero, in particular bit 30 (extended request) must be zero.

After the client sends the request message, the eStreamer service will immediately start sending event data if the requested event types have been enabled on the server side UI eStreamer configuration page.

Format of the JSON file

This example can be also found in the `json_request.json` file in the eStreamer SDK.

```
{ "Events":
```

This section specifies the requested fields from connection events. If this section were removed, the eStreamer server would not send any connection events.

```
{ "ConnectionEvent":
{ "FieldSetDef":
{ "OutputFieldSet":
["HeaderFieldSet", "ConnectionKeySet", "DetailFieldSet"] },
 "Fields":
["OutputFieldSet"] },
```

This section specifies the requested fields from intrusion events. If this section were removed, the eStreamer server would not send any intrusion events.

```
"IntrusionEvent":
{ "FieldSetDef":
{ "OutputFieldSet":
["HeaderFieldSet", "ConnectionKeySet", "DetailFieldSet", "Impact"] },
"Fields": ["OutputFieldSet"] },
```

This section specifies the requested fields from intrusion event packets. If this section were removed, the eStreamer server would not send any intrusion event packets.

```
"IntrusionPacket":
{ "FieldSetDef":
{ "OutputFieldSet":
["HeaderFieldSet", "DetailFieldSet"] },
"Fields": ["OutputFieldSet"] },
```

This section specifies the requested fields from file events. If this section were removed, the eStreamer server would not send any file events.

```
"FileEvent":
{ "FieldSetDef":
{ "OutputFieldSet":
["HeaderFieldSet", "ConnectionKeySet", "DetailFieldSet"] },
"Fields":
["OutputFieldSet"] } },
```

This section specifies the output format as described below.

```
"OutputFormat":
{ "Transform": "Text", "TransformConfig": "JSON" } }
```

In the `Events` section, specify a block for each event type that you would like the client to receive (only the three example types are supported: `ConnectionEvent`, `IntrusionEvent`, `IntrusionPacket`, and `FileEvent`). The `FieldSetDef` section for each event must specify an `OutputFieldSet`, which lists the fields or field sets which will be included in the events for that event type. The sample file only specifies field sets, but you can use any combination of field names and field sets.

The list of available fields for each event type, and the predefined field sets, can be found on the Firepower Management Center in the file `/etc/sf/EventHandler/EventCatalog/EventCatalog.json`. In the Fields section towards the end of the file, look for the desired event type (such as `IntrusionEvent`), then see the `Fields` and `FieldSetDef` blocks to see what is available for that event type.

The `OutputFormat` section has settings for the output. The `Transform` field is always `Text`, and you specify the output transformation format with the `TransformConfig` field. The example shows `JSON`, but you can also specify `CSV`. Other text formats are available, as well as `FlatBuffer`, but you will need to request documentation for these formats.

When JSON output is specified in `TransformConfig`, the output will contain name-value pairs for each requested field, except any fields which are irrelevant to the event are skipped (e.g. if you requested SSL fields, and an event did not use SSL, then the output will not contain those fields).

When CSV output is specified in `TransformConfig`, the output will contain the desired fields in the order listed in the configuration. If a field is not relevant to the event then the CSV will only contain a comma for that field. Do not use predefined field sets when requesting CSV because the field sets may change between versions, making the CSV incompatible.

Fully-Qualified Event Messages

Event messages are contained in bundles, as described in the eStreamer documentation for "Message Bundle Format", message type `4002`.

The client must acknowledge each received data bundle by sending a null message to the eStreamer server, indicating readiness to accept more data.

For all supported event types, the event data message starts with the binary header that is described in the eStreamer documentation for various event types, such as the "Intrusion Record Header". The only difference is that the data block format is the requested format (JSON, CSV, etc.). For quick reference the basic structure is:

```
<Header Version (1)>

<Message Type (3)>

<Message Length>

<Record Type (with optional Netmap ID when requested)>

<Record Length> <Timestamp (when request bit 23 is specified)>

<Reserved (when request bit 23 is specified)>

<Data>
```

Accepting Data from eStreamer

Note The eStreamer server does not keep a history of the events it sends. Your client application must check for duplicate events, which can inadvertently occur for a number of reasons. For example, when starting up a new streaming session, the time specified by the client as the starting point for the new session can have multiple messages, some of which may have been sent in the previous session and some of which were not. eStreamer sends all message that meet the specified request criteria. Your application should detect any resulting duplicates.

During periods of inactivity, eStreamer sends periodic null messages to the client to keep the connection open. If it receives an error message from the client or an intermediate host, it closes the connection.

eStreamer transmits requested data to the client differently, depending on the request mode.

Changing a Request

To change request parameters for an established session, the client must disconnect and request a new session.

Terminating Connections

The eStreamer server attempts to send an error message before closing the connection. For information on error messages, see Error Message Format.

The eStreamer server can close a client connection for the following reasons:

- Any time sending a message results in an error. This includes both event data messages and the null keep-alive message eStreamer sends during periods of inactivity.
- An error occurs while processing a client request.
- Client authentication fails (no error message is sent).
- eStreamer service is shutting down (no error message is sent).

Your client can close the connection to eStreamer server at any time and should attempt to use the error message format to notify the eStreamer server of the reason.