secevent
Internet-Draft
Intended status: Informational
Expires: February 11, 2018

M. Scurtescu
Google
A. Backman
Amazon

August 10, 2017

Management API for SET Event Streams
draft-scurtescu-secevent-event-stream-mgmt-api-00

Abstract

Security Event Token (SET) delivery requires event receivers to indicate to event transmitters the subjects about which they wish to receive events, and how they wish to receive them. This specification defines an HTTP API for a basic control plane that event transmitters can implement and event receivers may use to manage the flow of events from one to the other.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on February 11, 2018.

Table of Contents

1. Introduction

This specification defines an HTTP API to be implemented by Event Transmitters and that can be used by Event Receivers to query the Event Stream status, to add and remove subjects and to trigger verification.

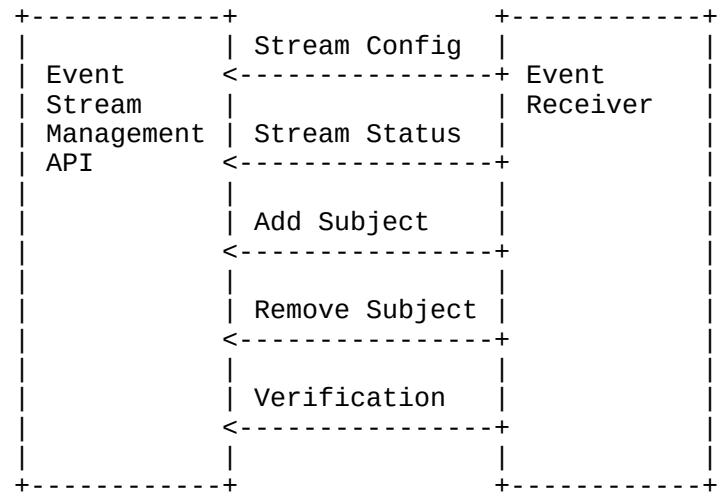

Figure 1: Event Stream Management API

How events are delivered and the structure of events are not in scope for this specification.

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Definitions

In addition to terms defined in [SET], this specification uses the following terms:

Subject Identifier Object
: A JSON object containing a set of one or more claims about a subject that when taken together uniquely identify that subject. This set of claims SHOULD be declared as an acceptable way to identify subjects of SETs by one or more specifications that profile [SET].

4. Event Stream Management

Event Receivers manage how they receive events, and the subjects about which they want to receive events over an Event Stream by making HTTP requests to endpoints in the Event Stream Management API.

The Event Stream Management API is implemented by the Event Transmitter and consists of the following endpoints:

Configuration Endpoint
: An endpoint used to read the Event Stream's current configuration.

Status Endpoint
: An endpoint used to read the Event Stream's current status.

Add Subject Endpoint
: An endpoint used to add subjects to an Event Stream.

Remove Subject Endpoint
: An endpoint used to remove subjects from an Event Stream.

Verification Endpoint
: An endpoint used to request the Event Transmitter transmit a Verification Event over the Event Stream.

An Event Transmitter MAY use the same URLs as endpoints for multiple streams, provided that the Event Transmitter has some mechanism through which they can identify the applicable Event Stream for any given request, e.g. from authentication credentials. The definition of such mechanisms is outside the scope of this specification.

4.1. Stream Configuration

An Event Stream's configuration is represented as a JSON object with the following properties:

aud
: A string containing an audience claim as defined in JSON Web Token (JWT) [RFC7519] that identifies the Event Receiver for the Event Stream. This property cannot be updated.

events
: OPTIONAL. An array of URIs identifying the set of events which MAY be delivered over the Event Stream. If omitted, Event Transmitters SHOULD make this set available to the Event Receiver via some other means (e.g. publishing it in online documentation).

delivery
: A JSON object containing a set of name/value pairs specifying configuration parameters for the SET delivery method. The actual delivery method is identified by the special key "delivery_method" with the value being a URI as defined in [DELIVERY].

min_verification_interval
An integer indicating the minimum amount of time in seconds that must pass in between verification requests. If an Event Receiver submits verification requests more frequently than this, the Event Transmitter MAY respond with a 429 status code. An Event Transmitter SHOULD NOT respond with a 429 status code if an Event Receiver is not exceeding this frequency.

status
A string indicating the current status of the event stream. It MUST have one of the following values:

enabled
The transmitter will transmit events over the stream, according to the stream's configured delivery method.

paused
The transmitter will not transmit events over the stream. The transmitter will hold any events it would have transmitted while paused, and will transmit them when the stream's status becomes "enabled".

disabled
The transmitter will not transmit events over the stream, and will not hold any events for later transmission.

4.1.1. Checking a Stream's Status

An Event Receiver checks the current status of an event stream by making an HTTP GET request to the stream's Status Endpoint. On receiving a valid request the Event Transmitter responds with a 200 OK response containing a [JSON] object with a single attribute "status", whose string value is the value of the stream's status.

The following is a non-normative example request to check an event stream's status:

```
GET /set/stream/status HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
```

Figure 2: Example: Check Stream Status Request

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "status": "enabled"
}
```

Figure 3: Example: Check Stream Status Response

4.1.2. Reading a Stream’s Configuration

An Event Receiver gets the current configuration of a stream by making an HTTP GET request to the Configuration Endpoint. On receiving a valid request the Event Transmitter responds with a 200 OK response containing a [JSON] representation of the stream’s configuration in the body.

The following is a non-normative example request to read an Event Stream’s configuration:

```
GET /set/stream HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
```

Figure 4: Example: Read Stream Configuration Request

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "aud": "http://www.example.com",
  "delivery": {
    "delivery_method": "urn:example:secevent:delivery:http_post",
    "url": "https://receiver.example.com/events"
  },
  "status": "enabled",
  "events": [
    "urn:example:secevent:events:type_1",
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3"
  ],
  "min_verification_interval": 60,
}
```

Figure 5: Example: Read Stream Configuration Response

Errors are signaled with HTTP staus codes as follows:

Code	Description
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to read the stream configuration
404	if there is no Event Stream configured for this Event Receiver

Table 1: Read Stream Configuration Errors

4.1.3. Updating a Stream’s Configuration

An Event Receiver updates the current configuration of a stream by making an HTTP POST request to the Configuration Endpoint. The POST body contains a {{!JSON} representation of the updated configuration. On receiving a valid request the Event Transmitter responds with a 200 OK response containing a [JSON] representation of the updated stream configuration in the body.

The full set of editable properties must be present in the POST body, not only the ones that are specifically intended to be changed. Missing properties SHOULD be interpreted as requested to be deleted. Event Receivers should read the configuration first, modify the [JSON] representation, then make an update request.

Properties that cannot be updated MAY be present, but they MUST match the expected value.

The following is a non-normative example request to read an Event Stream's configuration:

```
POST /set/stream HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "aud": "http://www.example.com",
  "delivery": {
    "delivery_method": "urn:example:secevent:delivery:http_post",
    "url": "https://receiver.example.com/events"
  },
  "status": "paused",
  "events": [
    "urn:example:secevent:events:type_1",
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3"
  ]
}
```

Figure 6: Example: Update Stream Configuration Request

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "aud": "http://www.example.com",
  "delivery": {
    "delivery_method": "urn:example:secevent:delivery:http_post",
    "url": "https://receiver.example.com/events"
  },
  "status": "paused",
  "events": [
    "urn:example:secevent:events:type_1",
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3"
  ]
}
```

Figure 7: Example: Update Stream Configuration Response

Errors are signaled with HTTP staus codes as follows:

Code	Description
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to update the stream configuration

Table 2: Update Stream Configuration Errors

4.2. Subjects

An Event Receiver can indicate to an Event Transmitter whether or not the receiver wants to receive events about a particular subject by "adding" or "removing" that subject to the Event Stream, respectively.

4.2.1. Adding a Subject to a Stream

To add a subject to an Event Stream, the Event Receiver makes an HTTP POST request to the Add Subject Endpoint, containing in the body a Subject Identifier Object identifying the subject to be added. On a successful response, the Event Transmitter responds with an empty 200 OK response.

The Event Transmitter MAY choose to silently ignore the request, for example if the subject has previously indicated to the transmitter that they do not want events to be transmitted to the Event Receiver. In this case, the transmitter MAY return an empty 200 OK response or an appropriate error code (See Security Considerations (Section 5)).

Errors are signaled with HTTP staus codes as follows:

Code	Description
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to add this particular subject
404	if the subject is not recognized by the Event Transmitter, the Event Transmitter may chose to stay silent in this case and respond with 200
429	if the Event Receiver is sending too many requests in a gvien amount of time

Table 3: Add Subject Errors

The following is a non-normative example request to add a subject to a stream, where the subject is identified by an OpenID Connect email claim:

```
POST /set/subjects:add HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "email": "example.user@example.com"
}
```

Figure 8: Example: Add Subject Request

The following is a non-normative example response to a successful request:

```
HTTP/1.1 200 OK
Server: transmitter.example.com
Cache-Control: no-store
Pragma: no-cache
```

Figure 9: Example: Add Subject Response

4.2.2. Removing a Subject

To remove a subject from an Event Stream, the Event Receiver makes an HTTP POST request to the Remove Subject Endpoint, containing in the body a Subject Identifier Object identifying the subject to be removed. On a successful response, the Event Transmitter responds with a 204 No Content response.

Errors are signaled with HTTP staus codes as follows:

Code	Description
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to remove this particular subject
404	if the subject is not recognized by the Event Transmitter, the Event Transmitter may chose to stay silent in this case and respond with 204
429	if the Event Receiver is sending too many requests in a gvien amount of time

Table 4: Remove Subject Errors

The following is a non-normative example request where the subject is identified by a phone_number claim:

```
POST /set/subjects:remove HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "phone_number": "+1 206 555 0123"
}
```

Figure 10: Example: Remove Subject Request

The following is a non-normative example response to a successful request:

```
HTTP/1.1 204 No Content
Server: transmitter.example.com
Cache-Control: no-store
Pragma: no-cache
```

Figure 11: Example: Remove Subject Response

4.3. Verification

In some cases, the frequency of event transmission on an Event Stream will be very low, making it difficult for an Event Receiver to tell the difference between expected behavior and event transmission failure due to a misconfigured stream. Event Receivers can request that a verification event be transmitted over the Event Stream, allowing the receiver to confirm that the stream is configured correctly upon successful receipt of the event. The acknowledgment of a Verification Event also confirms to the Event Transmitter that end-to-end delivery is working, including signature verification and encryption.

An Event Transmitter MAY send a Verification Event at any time, even if one was not requested by the Event Receiver.

4.3.1. Verification Event

The Verification Event is a standard SET with the following attributes:

event type
: The Event Type URI is: "urn:ietf:params:secevent:event-type:core:verification".

state
: OPTIONAL An opaque value provided by the Event Receiver when the event is triggered. This is a nested attribute in the event payload.

Upon receiving a Verification Event, the Event Receiver SHALL parse the SET and validate its claims. In particular, the Event Receiver SHALL confirm that the value for "state" is as expected. If the value of "state" does not match, an error response of "setData" SHOULD be returned (see Section 2.4 of [DELIVERY]).

In many cases, Event Transmitters MAY disable or suspend an Event Stream that fails to successfully verify based on the acknowledgement or lack of acknowledgement by the Event Receiver.

4.3.2. Triggering a Verification Event.

To request that a verification event be sent over an Event Stream, the Event Receiver makes an HTTP POST request to the Verification Endpoint, with a JSON object containing the parameters of the verification request, if any. On a successful request, the event transmitter responds with an empty 204 No Content response.

Verification requests have the following properties:

state
: OPTIONAL. An arbitrary string that the Event Transmitter MUST echo back to the Event Receiver in the verification event's payload. Event Receivers MAY use the value of this parameter to correlate a verification event with a verification request. If the verification event is initiated by the transmitter then this parameter MUST not be set.

A successful response from a POST to the Verification Endpoint does not indicate that the verification event was transmitted successfully, only that the Event Transmitter has transmitted the event or will do so at some point in the future. Event Transmitters MAY transmit the event via an asynchronous process, and SHOULD publish an SLA for verification event transmission times. Event Receivers MUST NOT depend on the verification event being transmitted synchronously or in any particular order relative to the current queue of events.

Errors are signaled with HTTP staus codes as follows:

Code	Description
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
429	if the Event Receiver is sending too many requests in a gvien amount of time

Table 5: Verification Errors

The following is a non-normative example request to trigger a verification event:

```
POST /set/verify HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
Content-Type: application/json; charset=UTF-8

{
  "state": "VGhpcyBpcyBhbiBleGFtcGxlIHN0YXRlIHZhbHVlLgo="
}
```

Figure 12: Example: Trigger Verification Request

The following is a non-normative example response to a successful request:

```
HTTP/1.1 204 No Content
Server: transmitter.example.com
Cache-Control: no-store
Pragma: no-cache
```

Figure 13: Example: Trigger Verification Response

And the following is a non-normative example of a verification event sent to the Event Receiver as a result of the above request:

```
{
  "jti": "123456",
  "iss": "https://transmitter.example.com",
  "aud": "receiver.example.com",
  "iat": "1493856000",
  "events": [
    "urn:ietf:params:secevent:event-type:core:verification" : {
      "state": "VGhpcyBpcyBhbiBleGFtcGxlIHN0YXRlIHZhbHVlLgo=",
    },
  ],
}
```

Figure 14: Example: Verification SET

5. Security Considerations

5.1. Subject Probing

It may be possible for an Event Transmitter to leak information about subjects through their responses to add subject requests. A 404 response may indicate to the Event Receiver that the subject does not exist, which may inadvertantly reveal information about the subject (e.g. that a particular individual does or does not use the Event Transmitter’s service).

Event Transmitters SHOULD carefully evaluate the conditions under which they will return error responses to add subject requests. Event Transmitters MAY return a 204 response even if they will not actually send any events related to the subject, and Event Receivers MUST NOT assume that a 204 response means that they will receive events related to the subject.

5.2. Information Harvesting

SETs may contain personally identifiable information (PII) or other non-public information about the event transmitter, the subject (of an event in the SET), or the relationship between the two. It is important for Event Transmitters to understand what information they are revealing to Event Receivers when transmitting events to them, lest the event stream become a vector for unauthorized access to private information.

Event Transmitters SHOULD interpret add subject requests as statements of interest in a subject by an Event Receiver, and ARE NOT obligated to transmit events related to every subject an Event Receiver adds to the stream. Event Transmitters MAY choose to transmit some, all, or no events related to any given subject and SHOULD validate that they are permitted to share the information contained within an event with the Event Receiver before transmitting the event. The mechanisms by which such validation is performed are outside the scope of this specification.

5.3. Malicious Subject Removal

A malicious party may find it advantageous to remove a particular subject from a stream, in order to reduce the Event Receiver’s ability to detect malicious activity related to the subject, inconvenience the subject, or for other reasons. Consequently it may be in the best interests of the subject for the Event Transmitter to continue to send events related to the subject for some time after the subject has been removed from a stream.

Event Transmitters MAY continue sending events related to a subject for some amount of time after that subject has been removed from the stream. Event Receivers MUST tolerate receiving events for subjects that have been removed from the stream, and MUST NOT report these events as errors to the Event Transmitter.

6. Normative References

[DELIVERY] "SET Token Delivery Using HTTP", n.d., <https://github.com/independentid/Identity-Events/blob/master/draft-hunt-secevent-delivery.txt>.

[JSON] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

[SET] "Security Event Token (SET)", n.d., <https://tools.ietf.org/html/draft-ietf-secevent-token-01>.

Authors' Addresses

Marius Scurtescu
Google

Email: mscurtescu@google.com

Annabelle Backman
Amazon

Email: richanna@amazon.com