Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 6, 2016

P. Hunt, Ed.
Oracle
M. Ansari
Cisco

April 4, 2016

Identity Event Subscription Protocol
draft-hunt-idevent-distribution-00

Abstract

This specification defines a registry to define methods to distribute identity events to subscribers. It includes a definition for publishers to use HTTP POST to push events to clients via web callback, and a method for subscribers to use HTTP GET to retrieve events in a feed queue.

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 October 6, 2016.

Copyright Notice

Table of Contents

1. Introduction and Overview

Many service providers have a requirement to co-ordinate state of entities and services between each other. Each service provider often tracks different information about entities and thus positive update commands such as HTTP POST or PATCH may not be possible as this would introduce complex error and signal requirements. In contrast, when one service provider notifies another of an event, the subscriber is free to take local action as it has access to the relevant local state information.

This specification defines a set of capabilities that can be used by publishers to distribute identity event tokens (see [idevent-token]) to subscribers. This specification defines a registry for profiling existing messaging protocols that may be used for event delivery by a particular subscriber. The specification also defines two methods HTTP POST and GET to deliver events.

The following diagram shows a typical Identity Feed Provider and its event notification Subscribers:

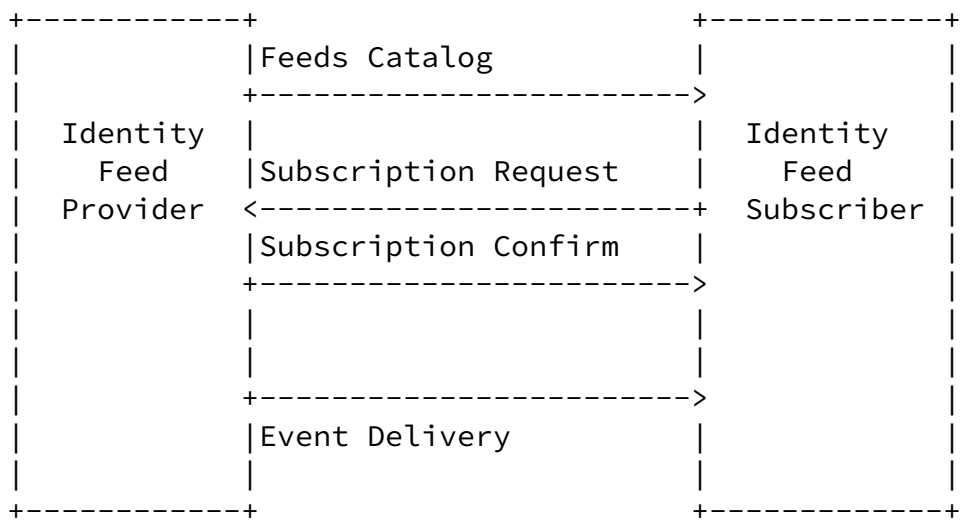

Figure 1: Subscription Management

An Identity feed provider may be directly integrated into a source service that generates events, or it may be a separate service entity that off-loads event distribution from the event generator to act as its publisher. For the purposes of this specification, while event distribution may be handled separately, this specification will consider the definition of those exchanges out of scope.

This specification addresses event delivery only. It is assumed that there are other protocols or administrative methods for providing event feeds catalogs and subscription management.

1.1. 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] . These keywords are capitalized when used to unambiguously specify requirements of the protocol or application features and behavior that affect the inter-operability and security of implementations. When these words are not capitalized, they are meant in their natural-language sense.

For purposes of readability examples are not URL encoded. Implementers MUST percent encode URLs as described in Section 2.1 of [RFC3986] .

Throughout this documents all figures MAY contain spaces and extra line-wrapping for readability and space limitations. Similarly, some URI's contained within examples, have been shortened for space and readability reasons.

1.2. Definitions

The following definitions are specific to Identity Event publishing:

Feed Provider The feed provider publishes events to be distributed to registered subscribers.

Event An event is an identify event [idevent-token] that is to be distributed to one or more registered subscribers. An event is

encapsulated as a JWT token and MAY be signed or encrypted using JWS/JWE for authentication and confidentiality reasons.

Feed A feed is a URI that describes the set of resources and events under which events may be issued. An interested client registers with the feed provider to subscribe to events associated with a feed.

Notification Mechanism A URI that describes the chosen event notification mechanism. When subscribing to a feed, a client may choose a specific mechanism by which it wishes to receive notification events. Examples of possible delivery mechanisms include:

Registered Callback - The feed provider will deliver events by using HTTP POST to a registered endpoint.

Polling - The subscriber will periodically poll the feed provider for one or more events by performing an HTTP GET to a specified URI (mailbox endpoint).

Platform/Mobile Notification Services (e.g. Apple Push Notification Service, Google Cloud Messaging, and Windows Notification Services). Future profiles that support delivery of SCIM events vis platform specific messaging services.

Subscriber A Subscriber registers to receive event notifications from a feed provider.

2. Event Notification Process

When an event occurs, the feed provider constructs a JWT based Identity Event token [idevent-token] that describes the event. The feed provider determines the feeds that the event should be published to, and determines which subscribers are effected. The process by which events are categorized and selected for subscribers is out of scope of this specification.

When an event is available for a subscriber, the feed provider

attempts to deliver the event based on the subscribers registered delivery mechanism. For example,

- The subscriber provided a web-callback endpoint, the publisher uses an HTTP/1.1 POST to the endpoint to deliver the event to the registered subscriber;

- For subscribers electing to poll for events, the feed publisher retains the events for a period of time or until the registered subscriber retrieves all pending events via HTTP/1.1 GET to the subscriber's assigned retrieval endpoint by the feed publisher; or,

- The subscriber elected to use an alternate delivery method (e.g. WebPUSH, Apple APNS, Google GMS), delivery is facilitated via the registered delivery profile for that method.

After an event is delivered to all subscribers, feed providers will not typically maintain event records or histories. As such, published events SHOULD be self-validating (e.g. signed).

If delivery to any particular subscriber has been delayed for an extended period of time, the feed provider MAY suspend the subscription and even stop maintaining outstanding events for the subscriber at its discretion and available resources. See subscription "state" in Section 4.2.

Upon receiving an event token (or tokens in the case of multiple events), the subscriber reads the token and validates it. Based on the content of the token, the subscriber decides what if any action it needs to take in response to the event. For example, in response to a SCIM event [idevent-scim] indicating a changed resource, the subscriber might perform a SCIM GET request (see Section 3.4 [RFC7644]) to the affected resource URI in order to confidentially obtain the current state of the affected resource. The receiver of the event then determines what action, if any, needs to be taken within the subscriber's domain.

The action a receiver takes may be substantially different than merely copying the action of the publisher. A single publisher event MAY trigger multiple receiver actions. For example, upon receiving notification that a user resource has been added to a group, the

receiver may first determine that the user does not exist in the subscriber's domain. The receiver translates the event into two actions. Retrieve the user (e.g. using SCIM GET) and then provisions the user locally. After enabling the user, the receiver then enables the user for the application associated with membership in the publisher's group.

3. Event Feeds

An event feed is service that provides a series of events made available that a client (known as the "subscriber") MAY subscribe to. A subscription contains the information about a particular client and their subscription to a particular "feedUri". The subscription metadata describes the client that has subscribed, the current delivery status indicating whether all events are delivered, pending, or whether delivery has failed. Subscription metadata also describes the method of event delivery and any associated configuration information (see Section 4.2).

3.1. Feed Types

A feed provider MAY define feeds based on a number of criteria. This specification does not specify or limit the basis for which a service provider defines a feed or how feed URIs should be specified. Some possible methods for defining feeds include:

By Resource Each resource might have its own event notification feed. For example, a User's mobile application may require notification of changes or rights defined in a SCIM User resource associated with the mobile user.

By Endpoint A feed might be defined by an endpoint where any event relating to a resource within an endpoint is delivered to a

subscriber. This type of feed is likely to have many notifications as the number of resources in an endpoint grows (e.g. a SCIM "/Users"). Typically only privileged partners would be allowed to use this type of feed. For example an enterprise wishes to be notified of all events to any of its Users assuming all Users within the endpoint are related to the subscribing enterprise.

By Filter A feed might define a collection of resources based on a filter that describes a set of matching criteria a resource may be included in a feed. Note that this type of feed may require extra processing by the service provider to determine if any particular resource event matches the filter criteria. It may also be difficult for the service provider to notify subscribers of Feed additions and deletions as these will occur dynamically based on the filter.

By Group For example, all resources within a SCIM Group could be used to define the resources within a particular feed. [TODO define a FEED Group extensions that define the attributes and events included within a particular Feed Group]

How feeds are defined or implemented is out of the scope of this specification. The above are examples about how feeds might be defined.

3.2. Feed Metadata

A feed description consists of the following singular attributes:

feedName
: A required string value containing a name for the feed. May be used in administrative user interfaces to assist subscribers in feed selection. The value MUST be unique within a given administrative domain. This is a required attribute.

feedUri
: An attribute of type "String" that is a unique URI identifying the feed. This attribute characteristic "mutability" is "immutable" and SHALL NOT change once assigned. This is a required attribute.

description
: A "String" attribute that describes the purpose of the feed in human readable form. This is an optional attribute.

type
: A "Reference" attribute that is one of the following canonical values:

resource Indicates that the feed is for events related to a

specific resource. In such cases, the value of the attribute "filter" is set to a specific resource URI or "/Me" .

endpoint Indicates that the feed is for all events that occur for resources within a specific endpoint. In such cases, "filter" is set to an endpoint container for a group of resources (e.g. "/Users").

filter Indicates that events for a feed will be selected based on events relating to the set of resources described by a filter. The value of the attribute "filter" is a SCIM filter that describes a condition that selects a set of resources that match before or after a resource state change.

group Indicates that events for a feed will be based on events relating to the set of resources listed in a SCIM Group. The value of the attribute "filter" is a URI that corresponds to a SCIM Group containing a set of members to be monitored.

hangText="publisher">Indicates a group whose definition is specific to the service provider publisher.The value of the attribute "filter" if used, is defined by the service provider.[[SHOULD THERE BE AN EXTENSION POINT?]]

filter
A String value containing a SCIM filter (see Section 3.4.2.2 [RFC7644]), a resource, or a SCIM endpoint URI. The contents of the value is indicated by the feed "type" attribute.

The following multi-valued attributes are defined:

events One or more String values that contain the Event URIs supported[[TBD]]. By default, all available events MAY be published.

deliveryModes One or more URIs representing the methods of delivery supported by the feed. By default, all delivery modes are supported.

4. Subscriptions

4.1. Overview

A subscription represents an agreement to deliver events from a specified feed of events from a feed provider to an individual subscriber entity. The method of delivery and the parameters for

delivery are specified a set of parameters called subscription metadata (see Section 4.2).

4.2. Subscription Metadata

A subscription is defined by the following metadata:

feedUri
: A string value containing the URI for a feed supported by the feed provider. It describes the content of the feed and MAY also be a resolvable URI where the feed meta data may be returned as a JSON object. REQUIRED.

deliveryUri
: A REQUIRED single-valued string which is a URI with a prefix of "urn:ietf:params:event:delivery". The following are defined in Section 5:

  urn:ietf:params:event:delivery:HTTP:webCallback
  : The feed provider delivers events using HTTP POST to a specified callback URI.

  urn:ietf:params:event:delivery:HTTP:poll
  : The subscriber will poll for events using HTTP GET.

subUri
: A URI representing the unique identifier for a single subscriber of a feed. It MAY also be an actual event polling endpoint which the subscriber MAY use to receive such as with "deliveryUri" method of "urn:ietf:params:event:delivery:HTTP:poll".

feedJwk
: AN OPTIONAL JSON Web Key (see [RFC7517]) that will be used to sign published events. If present, the subscriber can authenticate events relayed from the feed provider.

confidentialJwk
: An OPTIONAL subscriber provided public JSON Web Key (see [RFC7517]) that may be used by the feed provider to encrypt event messages for the subscriber.

subStatus
: An optional value which indicates the current status of a subscription:

  "on" - the default setting indicates the feed provider

processing events and will pass them to the subscriber.

"verify" - the subscription is pending verification. While in "verify", published events SHALL NOT be stored or delivered to the subscriber.

"paused" - indicates the feed provider is temporarily suspending delivery to subscriber. While in "paused" events SHOULD be retained and delivered when state returns to "on".

"off" - indicates that the subscription is no longer passing events. While in off mode, the subscription metadata is maintained, but new events are ignored and not processed or retained.

"fail" - Indicates that the feed provider was unable to deliver events to the subscriber for an extended period of time, or due to a call failure to the registered web call back URI. Unlike paused status, a failed subscription is no longer receiving events nor are they retained by the feed provider.

maxRetries
An OPTIONAL number indicating the maximum number of attempts to deliver an event. A value of '0' indicates there is no maximum. Upon reaching the maximum, the subscription "subStatus" is set to "failed" [[or "paused"?]].

maxDeliveryTime
An OPTIONAL number indicating the maximum amount of time an event MAY wait before delivery. Upon reaching the maximum, the subscription "subStatus" is set to "failed" [[or "paused"?]].

minDeliveryInterval
An OPTIONAL integer that represents the minimum interval in seconds between deliveries. A value of '0' indicates delivery should happen immediately. When delivery is a polling method (e.g. HTTP GET), it is the expected time between subscriber attempts. When in push mode (e.g. HTTP POST), it is the interval the server will wait before sending a new event or events.

4.3. Subscription Verification

In order to avoid ongoing communication issues and to minimize requirements for feed providers to maintain events indefinitely, a verification process is used to confirm that the requested event distribution endpoints are correct and that events may be successfully delivered. When a subscription is created or modified, the feed provider SHALL set the subscription state ("subStatus") to "verify" and send a test event message to the subscriber. If the event is delivered successfully, the subscription state MAY be turned

to "on". If however verification fails due to a hard failure, or the client fails to retrieve the event within a [reasonable?] period, the subscription status SHALL be set to "fail".

4.3.1. Verifying 'Push' Style Subscriptions

When using the WebCallback mode, or any other 'push'-style communication scheme, the verification process serves to verify that the identified endpoint consents to receiving events and is valid. This prevents a notification server from flooding an endpoint which did not actually request an event subscription.

To confirm a subscription, the feed provider SHALL send a verification event to the subscriber using the registered _deliveryUri_ mechanism. The event contains the following attributes:

eventUris Set with a value of "urn:ietf:params:event:event:verify".

iss Set to the URI of the feed publisher (see [idevent-token]).

aud MUST be set to a value that matches the subscription "feedUri" requested.

confirmChallenge A String value that the subscriber SHALL echo back in its response. See deliveryUri method profile for usage details.

exp A value that indicates the time the verification request will expire. Once expired, the server will set the subscription state to "fail".

If a confidential JWK was supplied, then the event SHOULD be encrypted with the provided key. Successful parsing of the message confirms that both the endpoint is valid including confirmation of keys.

A non-normative JSON representation of an event to be sent to a subscriber as a subscription confirmation. Note the event is not yet encoded as a JWT token.

```
{
  "jti": "4d3559ec67504aaba65d40b0363faad8",
  "eventUris":["urn:ietf:params:event:event:verify"],
  "iat": 1458496404,
  "iss": "https://scim.example.com",
  "aud":[
   "https://scim.example.com/Feeds/98d52461fa5bbc879593b7754",
   "https://scim.example.com/Feeds/5d7604516b1d08641d7676ee7"
  ],
  "confirmChallenge":"ca2179f4-8936-479a-a76d-5486e2baacd7",
  "exp": 1458497000

}
```

Upon receiving a subscription confirmation request, a confirming subscriber responds with a confirmation using the method described in the deliveryUri profile. The response includes a "challengeResponse" value. For example, depending on the deliveryUri profile used, the subscriber might respond with the value of "confirmChallenge" . For example, if the request is received via HTTP/1.1 POST, then the following HTTP response is used to confirm. A non-normative example

of the response is:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "challengeResponse":"ca2179f4-8936-479a-a76d-5486e2baacd7"
}
```

If the subscriber returns a non-matching value or an HTTP status other than a 200 series response, the subscription "state" SHALL be set to "fail". A declining subscriber MAY simply respond with any 400 series HTTP error (e.g. 404).

4.3.2. Verifying 'Polling' Style Subscriptions

For clients that use a subscription mode (e.g. "urn:ietf:params:scimnotify:api:messages:2.0:poll") that pick up events from a subscription endpoint, the client MAY confirm the subscription by simply reading the event using an HTTP GET at the endpoint specified by the attribute "subUri" in the subscription. Once the confirmation event has been retrieved, the service provider MAY mark the subscription as confirmed.

A non-normative example of a client, having previously subscribed, picking up the initial subscription confirmation message.

```
GET /Events/548b7c3f77c8bab33a4fef40/
Host: example.com
Accept: application/scim+json
Authorization: Bearer h480djs93hd8
Content-Length: ...
```

To which the event provider responds with the available events which SHOULD include a confirmation event (non-normative example):

```
{
  "schemas":["urn:ietf:params:scim:schemas:notify:2.0:Event"],
  "publisherUri":"https://scim.example.com",
  "feedUris":[
   "https://notify.example.com/Feeds/98d52461fa5bbc879593b7754"
```

```
    ],
    "type":"CONFIRMATION",
    "confirmChallenge":"ca2179f4-8936-479a-a76d-5486e2baacd7",
    "expires":""

  }
```

5. Event Delivery

5.1. Introduction to Event Delivery Methods

Each event delivery method SHALL have the following information:

Description
: The _deliveryUri_ URI value for the delivery method and a description of the method.

Subscription Verification Procedure
: The procedure that the configuration for a subscription is confirmed causing the subscription status to be set to "on".

Delivery Message Format
: A description of an event delivery message and how to locate the event token(s) as well as any additional signaling parameters.

Delivery Procedure
: The protocol procedure for a delivery request (push or poll), and the expected successful response.

Failure Conditions

: A description of the failure conditions that might occur and the impact on the subscriptions operational status ("subStatus") if any.

Multi-Event Message Considerations
: A description of any special considerations when passing multiple events in a single delivery.[[is this needed?]]

5.2. HTTP Web Callback Method

5.2.1. Description

This method allows a feed provider to use HTTP POST to deliver events to a registered web callback URL. The "deliveryUri" value for this method is "urn:ietf:params:event:delivery:HTTP:webCallback".

Depending on the settings for the subscription metadata, this delivery method is capable of delivering multiple signed events in a single delivery.

5.2.2. Delivery Message Format

The content-type for this method is "application/json" and consists of a JSON object containing the following attributes:

eventTkns
: A multi-valued String with each value containing an identity event token ([idevent-token]). Each value MAY represent an unsigned, signed, or encrypted token. At least one value MUST be present in a delivery request.

eventCnt
: An integer representing the number of events present in the message. REQUIRED.

eventPend
: An boolean indicating more deliveries are pending. The default value is false.

invalidEvents
: An optional multi-valued set of JSON objects containing events that could not be accepted or failed. Used in a delivery response, a subscriber MAY return an event that failed to validate or was unparseable.


```
{
"eventTkns":[
  "eyJhbGciOiJub25lIn0
  .
```

```
    eyJwdWJsaXNoZXJVcmkiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20iLCJmZWV
    kVXJpcyI6WyJodHRwczovL2podWIuZXhhbXBsZS5jb20vRmVlZHMvOThkNTI0Nj
    FmYTViYmM4Nzk1OTNiNzc1NCIsImh0dHBzOi8vamh1Yi5leGFtcGxlLmNvbS9GZ
    WVkcy81ZDc2MDQ1MTZiMWQwODY0MWQ3Njc2ZWU3Il0sInJlc291cmNlVXJpcyI6
    WyJodHRwczovL3NjaW0uZXhhbXBsZS5jb20vVXNlcnMvNDRmNjE0MmRmOTZiZDZ
    hYjYxZTc1MjFkOSJdLCJldmVudFR5cGVzIjpbIkNSRUFURSJdLCJhdHRyaWJ1dG
    VzIjpbImlkIiwibmFtZSIsInVzZXJOYW1lIiwicGFzc3dvcmQiLCJlbWFpbHMiX
    SwidmFsdWVzIjp7ImVtYWlscyI6W3sidHlwZSI6Indvcmsi LCJ2YWx1ZSI6Impk
    b2VAZXhhbXBsZS5jb20ifV0sInBhc3N3b3JkIjoibm90NHUybm8iLCJ1c2VyTmF
    tZSI6Impkb2UiLCJpZCI6IjQ0ZjYxNDJkZjk2YmQ2YWI2MWU3NTIxZDkiLCJuYW
    1lIjp7ImdpdmVuTmFtZSI6IkpvaG4iLCJmYW1pbHlOYW1lIjoiRG9lIn19fQ
    ."],
  "eventCnt":1,
  "eventPend":false
  }
```


Example Web Callback POST Message

5.2.3. Subscription Verification

See Section 4.3.1.

5.2.4. Delivery Procedure

To deliver an event, the publisher generates an event delivery message and uses HTTP POST to the registered endpoint.

```
POST /Events  HTTP/1.1
Host: notify.example.com
Accept: application/json
Content-Type: application/json
{
"eventTkns":[
  "eyJhbGciOiJub25lIn0
  .
  eyJwdWJsaXNoZXJVcmkiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20iLCJmZWV
  kVXJpcyI6WyJodHRwczovL2podWIuZXhhbXBsZS5jb20vRmVlZHMvOThkNTI0Nj
  FmYTViYmM4Nzk1OTNiNzc1NCIsImh0dHBzOi8vamh1Yi5leGFtcGxlLmNvbS9GZ
  WVkcy81ZDc2MDQ1MTZiMWQwODY0MWQ3Njc2ZWU3Il0sInJlc291cmNlVXJpcyI6
  WyJodHRwczovL3NjaW0uZXhhbXBsZS5jb20vVXNlcnMvNDRmNjE0MmRmOTZiZDZ
  hYjYxZTc1MjFkOSJdLCJldmVudFR5cGVzIjpbIkNSRUFURSJdLCJhdHRyaWJ1dG
  VzIjpbImlkIiwibmFtZSIsInVzZXJOYW1lIiwicGFzc3dvcmQiLCJlbWFpbHMiX
  SwidmFsdWVzIjp7ImVtYWlscyI6W3sidHlwZSI6Indvcmsi LCJ2YWx1ZSI6Impk
  b2VAZXhhbXBsZS5jb20ifV0sInBhc3N3b3JkIjoibm90NHUybm8iLCJ1c2VyTmF
  tZSI6Impkb2UiLCJpZCI6IjQ0ZjYxNDJkZjk2YmQ2YWI2MWU3NTIxZDkiLCJuYW
  1lIjp7ImdpdmVuTmFtZSI6IkpvaG4iLCJmYW1pbHlOYW1lIjoiRG9lIn19fQ
  .",
  "eyJhbGciOiJub25lIn0
  .
  eyJwdWJsaXNoZXJVcmkiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20iLCJmZWV
  kVXJpcyI6WyJodHRwczovL2podWIuZXhhbXBsZS5jb20vRmVlZHMvOThkNTI0Nj
  FmYTViYmM4Nzk1OTNiNzc1NCIsImh0dHBzOi8vamh1Yi5leGFtcGxlLmNvbS9GZ
  WVkcy81ZDc2MDQ1MTZiMWQwODY0MWQ3Njc2ZWU3Il0sInJlc291cmNlVXJpcyI6
  WyJodHRwczovL3NjaW0uZXhhbXBsZS5jb20vVXNlcnMvNDRmNjE0MmRmOTZiZDZ
  hYjYxZTc1MjFkOSJdLCJldmVudFR5cGVzIjpbIkNSRUFURSJdLCJhdHRyaWJ1dG
  VzIjpbImlkIiwibmFtZSIsInVzZXJOYW1lIiwicGFzc3dvcmQiLCJlbWFpbHMiX
  SwidmFsdWVzIjp7ImVtYWlscyI6W3sidHlwZSI6Indvcmsi LCJ2YWx1ZSI6Impk
  b2VAZXhhbXBsZS5jb20ifV0sInBhc3N3b3JkIjoibm90NHUybm8iLCJ1c2VyTmF
  tZSI6Impkb2UiLCJpZCI6IjQ0ZjYxNDJkZjk2YmQ2YWI2MWU3NTIxZDkiLCJuYW
  1lIjp7ImdpdmVuTmFtZSI6IkpvaG4iLCJmYW1pbHlOYW1lIjoiRG9lIn19fQ
  ."
  ],
"eventCnt":2
}
```

Example Web Callback POST Request

In response, if the event token is validated, the server SHALL indicate successful submission by responding with:

```
HTTP/1.1 202 Accepted
```

or to indicate errors it MAY respond with

```
HTTP/1.1 202 Accepted
Content-Type: application/json

{
  "invalidEvents":[
    {"err":"duplicate",
     "description":"Event already received. Ignored.",
     "value":"eyJhbGciOiJub25lIn0
  .
  eyJwdWJsaXNoZXJVcmkiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20iLCJmZWV
  kVXJpcyI6WyJodHRwczovL2podWIuZXhhbXBsZS5jb20vRmVlZHMvOThkNTI0Nj
  FmYTViYmM4Nzk1OTNiNzc1NCIsImh0dHBzOi8vamh1Yi5leGFtcGxlLmNvbS9GZ
  WVkcy81ZDc2MDQ1MTZiMWQwODY0MWQ3Njc2ZWU3Il0sInJlc291cmNlVXJpcyI6
  WyJodHRwczovL3NjaW0uZXhhbXBsZS5jb20vVXNlcnMvNDRmNjE0MmRmOTZiZDZ
  hYjYxZTc1MjFkOSJdLCJldmVudFR5cGVzIjpbIkNSRUFURSJdLCJhdHRyaWJ1dG
  VzIjpbImlkIiwibmFtZSIsInVzZXJOYW1lIiwicGFzc3dvcmQiLCJlbWFpbHMiX
  SwidmFsdWVzIjp7ImVtYWlscyI6W3sidHlwZSI6Indvcmsi LCJ2YWx1ZSI6Impk
  b2VAZXhhbXBsZS5jb20ifV0sInBhc3N3b3JkIjoibm90NHUybm8iLCJ1c2VyTmF
  tZSI6Impkb2UiLCJpZCI6IjQ0ZjYxNDJkZjk2YmQ2YWI2MWU3NTIxZDkiLCJuYW
  1lIjp7ImdpdmVuTmFtZSI6IkpvaG4iLCJmYW1pbHlOYW1lIjoiRG9lIn19fQ
  ."
    }
  ]
}
```

Since the normal operation of the Feed Provider is to forward events to registered subscribers, the Feed Provider is not obligated to inform the publisher of a permanent event URI that was created. Servers MAY allow HTTP clients to check for undelivered events by performing a GET against the same endpoint as the Event submission endpoint described above.

5.2.5. Failure Conditions

[[TO BE COMPLETED]]

5.3. HTTP Polling

5.3.1. Description

This method allows a subscriber to use HTTP GET to poll for events to the "subUri" provided to the subscriber by the publisher. The "deliveryUri"value for this method is "urn:ietf:params:event:delivery:HTTP:poll".

Depending on the settings for the subscription metadata, this delivery method is capable of delivering multiple signed events in a single delivery poll request.

5.3.2. Delivery Message Format

The delivery message is the same as Section 5.2.2.

5.3.3. Subscription Verification

Subscription verification is described in Section 4.3.2.

5.3.4. Delivery Procedure

To deliver an event, the publisher places the event in a queue/buffer associated with the client subscription that will be requested at some future time by the subscriber using the URI specified in "subUri".

To pick up any events, the subscriber issues an HTTP GET to the URI provided by the event publisher in "subUri".

In response to the HTTP GET request, the feed publisher responds with a body that corresponds to the event delivery message format (see Section 5.2.2).

An example polling request by a subscriber. The example has been formatted for readability:

```
GET /subscriber/66444423ab24430fb06cf0de1ab75247
   Host: pub.example.com
   Accept: application/json
   Authorization: Bearer h480djs93hd8
```

An example poll response (formatted for readability):

```
HTTP/1.1 200 OK
Content-Type: application/json
Location:
  https://pub.example.com/subscriber/66444423ab24430fb06cf0de1ab75247
{
"eventTkns":[
  "eyJhbGciOiJub25lIn0
  .
  eyJwdWJsaXNoZXJVcmkiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20iLCJmZWV
  kVXJpcyI6WyJodHRwczovL2podWIuZXhhbXBsZS5jb20vRmVlZHMvOThkNTI0Nj
  FmYTViYmM4Nzk1OTNiNzc1NCIsImh0dHBzOi8vamh1Yi5leGFtcGxlLmNvbS9GZ
  WVkcy81ZDc2MDQ1MTZiMWQwODY0MWQ3Njc2ZWU3Il0sInJlc291cmNlVXJpcyI6
  WyJodHRwczovL3NjaW0uZXhhbXBsZS5jb20vVXNlcnMvNDRmNjE0MmRmOTZiZDZ
  hYjYxZTc1MjFkOSJdLCJldmVudFR5cGVzIjpbIkNSRUFURSJdLCJhdHRyaWJ1dG
  VzIjpbImlkIiwibmFtZSIsInVzZXJOYW1lIiwicGFzc3dvcmQiLCJlbWFpbHMiX
  SwidmFsdWVzIjp7ImVtYWlscyI6W3sidHlwZSI6Indvcmsi LCJ2YWx1ZSI6Impk
  b2VAZXhhbXBsZS5jb20ifV0sInBhc3N3b3JkIjoibm90NHUybm8iLCJ1c2VyTmF
  tZSI6Impkb2UiLCJpZCI6IjQ0ZjYxNDJkZjk2YmQ2YWI2MWU3NTIxZDkiLCJuYW
  1lIjp7ImdpdmVuTmFtZSI6IkpvaG4iLCJmYW1pbHlOYW1lIjoiRG9lIn19fQ
  .",
  "eyJhbGciOiJub25lIn0
  .
  eyJwdWJsaXNoZXJVcmkiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20iLCJmZWV
  kVXJpcyI6WyJodHRwczovL2podWIuZXhhbXBsZS5jb20vRmVlZHMvOThkNTI0Nj
```

```
  FmYTViYmM4Nzk1OTNiNzc1NCIsImh0dHBzOi8vamh1Yi5leGFtcGxlLmNvbS9GZ
  WVkcy81ZDc2MDQ1MTZiMWQwODY0MWQ3Njc2ZWU3Il0sInJlc291cmNlVXJpcyI6
  WyJodHRwczovL3NjaW0uZXhhbXBsZS5jb20vVXNlcnMvNDRmNjE0MmRmOTZiZDZ
  hYjYxZTc1MjFkOSJdLCJldmVudFR5cGVzIjpbIkNSRUFURSJdLCJhdHRyaWJ1dG
  VzIjpbImlkIiwibmFtZSIsInVzZXJOYW1lIiwicGFzc3dvcmQiLCJlbWFpbHMiX
  SwidmFsdWVzIjp7ImVtYWlscyI6W3sidHlwZSI6Indvcmsi LCJ2YWx1ZSI6Impk
  b2VAZXhhbXBsZS5jb20ifV0sInBhc3N3b3JkIjoibm90NHUybm8iLCJ1c2VyTmF
  tZSI6Impkb2UiLCJpZCI6IjQ0ZjYxNDJkZjk2YmQ2YWI2MWU3NTIxZDkiLCJuYW
  1lIjp7ImdpdmVuTmFtZSI6IkpvaG4iLCJmYW1pbHlOYW1lIjoiRG9lIn19fQ
  ."
  ],
"eventCnt":2
}
```


[[TO BE DISCUSSED: Should the subscriber be able to request events based on an event id (e.g. since), by date, etc. How does a client know it got them all? Should we use ETags to signal whether there are new events?]]

5.3.5. Failure Conditions

[[TO BE DISCUSSED: The polling mode provides no way for a subscriber to indicate parsing validation errors directly in response to a delivery. When errors occur, subscriber administrators must re-confirm the subscription configuration.]]

5.4. Push Notification Extensions

[[TO BE COMPLETED]]

6. Security Considerations

The synchronizing of User passwords between administrative domains is to be handled with great care. From a security perspective the re-use of passwords across service providers is to be highly discouraged. However, in the enterprise user experience, if the perception of the user is that service providers from multiple domains are part of a single corporate application environment, then

password synchronization MAY be appropriate as part of an overall identity management and governance mechanism.

[TO BE COMPLETED]

7. IANA Considerations

7.1. SCIM Event Notification Mechanism Registry

TODO: Registration for Notification Mechanisms

7.2. SCIM Event Type Registry

TODO: Registration of Event Types

8. References

8.1. Normative References

[idevent-token] Oracle Corporation, "Identity Event Token (work in progress)".

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[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>.

[RFC7643] Hunt, P., Ed., Grizzle, K., Wahlstroem, E., and C. Mortimore, "System for Cross-domain Identity Management: Core Schema", RFC 7643, DOI 10.17487/RFC7643, September 2015, <http://www.rfc-editor.org/info/rfc7643>.

[RFC7644] Hunt, P., Ed., Grizzle, K., Ansari, M., Wahlstroem, E., and C. Mortimore, "System for Cross-domain Identity Management: Protocol", RFC 7644, DOI 10.17487/RFC7644, September 2015, <http://www.rfc-editor.org/info/rfc7644>.

8.2. Informative References

[I-D.ietf-webpush-protocol]
Thomson, M., Damaggio, E., and B. Raymor, "Generic Event Delivery Using HTTP Push", draft-ietf-webpush-protocol-02 (work in progress), November 2015.

[idevent-scim]
Oracle Corporation, "SCIM Event Extensions (work in progress)".

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <http://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <http://www.rfc-editor.org/info/rfc7517>.

Appendix A. Contributors

Appendix B. Acknowledgments

The editor would like to thank the participants in the the SCIM

working group for their support of this specification.

Appendix C. Change Log

Draft 00 - PH - First Draft

Authors' Addresses

Phil Hunt (editor)
Oracle Corporation

Email: phil.hunt@yahoo.com

Morteza Ansari
Cisco

Email: morteza.ansari@cisco.com