SET Token Delivery Using HTTP

Marius Scurtrescu, Google
IETF99 Prague
July 2017
Overview

The specification defines:

● how SETs can be delivered to a receiver
  ○ push initiated by transmitter
  ○ poll initiated by receiver

● verification process to test an Event Stream

draft-hunt-secevent-delivery-00
Definitions

● Identity Provider
  ○ explicit: a service provider that issues authentication assertions
  ○ implicit: service providers that manage personal identifiers used in recovery scenarios by Relying Parties (i.e. email or phone number)

● Relying Party
  ○ explicit: a service provider that accepts authentication assertions
  ○ implicit: service providers that use a personal identifier managed by another provider

● Event Transmitter - a service provider that delivers SETs
● Event Receiver - a service provider that receives SETs
● Event Stream - a defined location and distribution method through which an Event Transmitter sends message to an Event Receiver
Event Delivery Process

- how SETs are defined and how SETs are assigned to Event Streams is out of scope
- when a SET is available in an Event Stream, the delivery is determined by the Event Receiver's registered mechanism:
  - push: HTTP POST to the Event Receiver endpoint
  - poll: the event is queued up in a buffer so the Event Receiver can poll using HTTP POST
  - another method not defined in this specification
- the Event Receiver must acknowledge receipt to the Event Transmitter
- after an acknowledged delivery the Event Transmitter is not required to maintain SETs
Push Delivery

- The Event Transmitter uses HTTP POST to deliver SETs to a configured callback URI
- request HTTP Content-Type: application/secevent+jwt
- Accept header: application/json
- a single SET per request
- if the SET is accepted then the response should be 202 (Accepted)
- in case of an error the Event Receiver may respond with an appropriate HTTP status code
POST /Events HTTP/1.1
Host: notify.examplerp.com
Accept: application/json
Authorization: Bearer h480djs93hd8
Content-Type: application/secevent+jwt
eyJhbGciOiJub25lIn0.
eyJwdsJWsaXNoZXJVcmkiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20iJiNjaW0uZXhhbXBsZS5jb20iLCJmZWVmdWVkJyI6WyJodHRwczovL2podWIuZXhhbXBsZS5jb20vRmV1ZHMyIn0sInBhc3N3b3JkIjoibm90NHUybm8iLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2UiLCJpZCI6Impkb2U
Push Deliver - Response Examples

Success:

HTTP/1.1 202 Accepted

Error:

HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "err":"dup",
   "description":"SET already received. Ignored."
}
Polling Delivery

- Event Receiver uses HTTP POST to both acknowledge SETs and to receive more SETs
- request & response HTTP Content-Type: application/json
- multiple SETs per response
- request consists of polling parameters, in JSON format
Polling Delivery - Request Attributes

Processing Parameters

- **maxEvents** - maximum number of SETs that should be returned
  - 0 (zero) means ack only request
- **returnImmediately** - false (the default) requests a long poll

SET Ack Parameters:

- **ack** - array of strings, each the "jti" of a successfully received SET
- **setErrs** - object with "jti" keys and "err"+"description" nested values
Polling Delivery - Response Attributes

- sets - object with "jti" keys and encoded SETs as values
- moreAvailable - boolean indicating that more SETs are available
Polling Delivery - Variations

1. Poll Only - no previous SETs to acknowledge
2. Acknowledge Only - maxEvents set to 0 and "ack" and/or "setErrs" present
3. Combined Acknowledge and Poll
Polling Delivery - Combined Request Example

POST /Events HTTP/1.1

Host: notify.exampleidp.com
Content-Type: application/json
Authorization: Bearer h480djs93hd8

{
    "ack": ["4d59ec67504aaba65d40b0363faad8", "3d0c3797584bd193bd0fb1bd4e7d30"],
    "setErrs": {
        "4d3559ec67504aaba65d40b03faad8": {
            "err": "jwtAud",
            "description": "The audience value was incorrect."
        }
    },
    "returnImmediately": false
}
Polling Delivery - Response Example

HTTP/1.1 200 OK
Content-Type: application/json
Location: https://notify.exampleidp/Events

{
    "sets":{
        "4d3559ec67504aaba65d40b0363faad8": "eyJhbGciOiJub25lIn0.
            2ZW50OmNyZWF0ZSI6eyJyZWYiOiJodHRwczovL3NjaW0uZXhhbXBsZS5jb20vVXNlcn
            W11IiwidXNlck5hbWUiLCJwYXNzd29yZCI6ImVtYWlscyJdfX19.",
        "3d0c3cf797584bd193bd0fb1bd4e7d30": "eyJhbGciOiJub25lIn0.
            eyJqdGkiOiIzZDBjM2NmNzk3NTg0YmQxOTNiZDBmYjFiZDRlN2QzMCIsImVtYWls
            cyJdfX19.",
    }
}
## SET Errors 1/2

<table>
<thead>
<tr>
<th>err</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>json</td>
<td>Invalid JSON object</td>
</tr>
<tr>
<td>jwtParse</td>
<td>Invalid or unparsable JWT or JSON structure</td>
</tr>
<tr>
<td>jwtHdr</td>
<td>An invalid JWT header was detected</td>
</tr>
<tr>
<td>jwtCrypto</td>
<td>Unable to parse due to unsupported algorithm</td>
</tr>
<tr>
<td>jws</td>
<td>Signature was not validated</td>
</tr>
<tr>
<td>jwe</td>
<td>Unable to decrypt JWE encoded data</td>
</tr>
<tr>
<td>jwtAud</td>
<td>Invalid audience value</td>
</tr>
<tr>
<td>jwtLss</td>
<td>Issuer not recognized</td>
</tr>
<tr>
<td>setType</td>
<td>An unexpected Event type was received</td>
</tr>
</tbody>
</table>
## SET Errors 2/2

<table>
<thead>
<tr>
<th>err</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setParse</td>
<td>Invalid structure was encountered such as an inability to parse or an incomplete set of Event claims</td>
</tr>
<tr>
<td>setData</td>
<td>SET event claims incomplete or invalid</td>
</tr>
<tr>
<td>dup</td>
<td>A duplicate SET was received and has been ignored</td>
</tr>
</tbody>
</table>
Event Stream Verification

- Event Receiver initiates a request to verify the stream
  - it provides "confirm" and "nonce" values
- Event Transmitter delivers Verify Event
Event Stream Verification - Example Event

```
{
    "jti": "4d3559ec67504aaba65d40b0363faad8",
    "iat": 1458496404,
    "iss": "https://scim.example.com",
    "exp": 1458497000,
    "aud": ["https://event.example.com/Feeds/98d52461fa5bbc879593b7754"],
    "events": {
        "[[this RFC URL]]#verify": {
            "confirm": "ca2179f4-8936-479a-a76d-5486e2baacd7",
            "nonce": "1668c993e95849869e4b3506cccdf9bf"
        }
    }
}
```
Authentication and Authorization

SET Delivery depends on TLS and/or standard HTTP authentication and authorization schemes.

For example:

- TLS Client Authentication
- Bearer Tokens
- Basic Authentication
- SET Payload Authentication
Q & A