Network Working Group
Internet-Draft
Intended status: Informational
Expires: 25 April 2019

K. Dunglas
Les-Tilleuls.coop

22 October 2018

The Mercure Protocol
draft-dunglas-mercure-02

Abstract

Mercure is a protocol allowing to push data updates to web browsers and other HTTP clients in a fast, reliable and battery-efficient way. It is especially useful to publish real-time updates of resources served through web APIs, to reactive web and mobile apps.

Status of This Memo

Copyright Notice

Table of Contents

1. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

* Topic: An HTTP [RFC7230] or HTTPS [RFC2818] topic URL. The unit to which one can subscribe to changes.

* Publisher: An owner of a topic. Notifies the hub when the topic feed has been updated. As in almost all pubsub systems, the publisher is unaware of the subscribers, if any. Other pubsub systems might call the publisher the "source". Typically a website or a web API, but can also be a web browser.

* Subscriber: A client application that subscribes to real-time updates of topics. Typically a Progressive Web App or a Mobile App, but can also be a server.

* Target: A subscriber, or a group of subscribers. A publisher is able to securely dispatch updates to specific targets. Using an HTTP [RFC7230] or HTTPS [RFC2818] URL to identify targets is RECOMMENDED.

* Hub: A server that handles subscription requests and distributes the content to subscribers when the corresponding topics have been updated (a Hub implementation is provided in this repository). Any hub MAY implement its own policies on who can use it.

2. Discovery

If the publisher is a server, it SHOULD advertise the URL of one or more hubs to the subscriber, allowing it to receive live updates when topics are updated. If more than one hub URL is specified, it is expected that the publisher notifies each hub, so the subscriber MAY subscribe to one or more of them.

The publisher SHOULD include at least one Link Header [RFC5988] with "rel=mercure" (a hub link header). The target URL of these links MUST be a hub implementing the Mercure protocol.

Note: this relation type has not been registered yet [RFC5988]. During the meantime, the relation type "https://git.io/mercure" can be used instead.

The publisher MAY provide the following target attributes in the Link headers:

* "last-event-id": the globally unique identifier of the last event dispatched by the publisher at the time of the generation of this resource. If provided, it MUST be passed to the hub through a query parameter called "Last-Event-ID" and will be used to ensure that possible updates having been made during between the resource generation time and the connection to the hub are not lost. See section #Re-Connection-and-State-Reconciliation. If this attribute is provided, the publisher MUST always set the "id" parameter when sending updates to the hub.

* "content-type": the content type of the updates that will pushed by the hub. If omited, the subscriber MUST assume that the content type will be the same than the one of the original resource. Setting the "content-type" attribute is especially useful to hint that partial updates will be pushed, using formats such as JSON Patch [RFC6902] or JSON Merge Patch [RFC7386].

* "key-set=<JWKS>": the key(s) to decrypt updates encoded in the JWKS (JSON Web Key Set) format (see the Encryption section).

All these attributes are optional.

The publisher MAY also include one Link Header [RFC5988] with "rel=self" (the self link header). It SHOULD contain the canonical URL for the topic to which subscribers are expected to use for subscriptions. If the Link with "rel=self" is ommitted, the current URL of the resource MUST be used as fallback.

Minimal example:

```
GET /books/foo.jsonld HTTP/1.1
Host: example.com

HTTP/1.1 200 Ok
Content-type: application/ld+json
Link: <https://example.com/hub>; rel="mercure"

{"@id": "/books/foo.jsonld", "foo": "bar"}
```

Links embedded in HTML or XML documents (as defined in the WebSub recommendation) MAY also be supported by subscribers.

Note: the discovery mechanism described in this section is strongly inspired from the one specified in the WebSub recommendation (https://www.w3.org/TR/websub/#discovery).

3. Subscriptions

The subscriber subscribes to an URL exposed by a hub to receive updates of one or many topics. To subscribe to updates, the client opens an HTTPS connection following the Server-Sent Events specification (https://html.spec.whatwg.org/multipage/server-sent-events.html) to the hub's subscription URL advertised by the Publisher. The "GET" HTTP method must be used. The connection SHOULD use HTTP/2 to leverage mutliplexing and other advanced features of this protocol.

The subscriber specifies the list of topics to get updates for by using one or several query parameters named "topic". The value of these query parameters MUST be URI templates [RFC6570].

Note: an URL is also a valid URI template.

The protocol doesn't specify the maximum number of "topic" parameters that can be sent, but the hub MAY apply an arbitrary limit.

The EventSource JavaScript interface (https://html.spec.whatwg.org/multipage/server-sent-events.html#the-eventsource-interface) MAY be used to establish the connection. Any other appropriate mechanism including but not limited to readable streams (https://developer.mozilla.org/en-US/docs/Web/API/Streams_API/Using_readable_streams) and XMLHttpRequest (https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest) (used by popular polyfills) MAY also be used.

The hub sends updates concerning all subscribed resources matching the provided URI templates and the provided targets (see section #Authorization). If no targets are specied, the update is dispatched to all subscribers. The hub MUST send these updates as text/event-stream compliant events (https://html.spec.whatwg.org/multipage/server-sent-events.html#sse-processing-model).

The "data" property MUST contain the new version of the topic. It can be the full resource, or a partial update by using formats such as JSON Patch "@RFC6902" or JSON Merge Patch "@RFC7386".

All other properties defined in the Server-Sent Events specification MAY be used and SHOULD be supported by hubs.

The resource SHOULD be represented in a format with hypermedia capabilities such as JSON-LD [W3C.REC-json-ld-20140116], Atom [RFC4287], XML [W3C.REC-xml-20081126] or HTML [W3C.REC-html52-20171214].

Web Linking [RFC5988] SHOULD be used to indicate the IRI of the resource sent in the event. When using Atom, XML or HTML as serialization format for the resource, the document SHOULD contain a "link" element with a "self" relation containing the IRI of the resource. When using JSON-LD, the document SHOULD contain an "@id" property containing the IRI of the resource.

Example:

```
   // The subscriber subscribes to updates for the https://example.com/foo
topic
   // and to any topic matching https://example.com/books/{name}
   const url = new URL('https://example.com/hub');
   url.searchParams.append('topic', 'https://example.com/foo');
   url.searchParams.append('topic', 'https://example.com/bar/{id}');

   const eventSource = new EventSource(url);

   // The callback will be called every time an update is published
   eventSource.onmessage = function ({data}) {
       console.log(data);
   };
```

The hub MAY require that the subscribers are authorized to receive any update.

4. Publication

The publisher send updates by issuing "POST" HTTPS requests on the hub URL. When it receives an update, the hub dispatches it to subsribers using the established server-sent events connections.

An application CAN send events directly to the subscribers, without using an external hub server, if it is able to do so. In this case, it *MAY NOT* implement the endpoint to publish updates.

The request MUST be encoded using the "application/x-www-form-urlencoded" format and contains the following data:

- "topic": IRIs of the updated topic. If this key is present several times, the first occurence is considered to be the canonical URL of the topic, and other ones are considered to be alternate URLs. The hub MUST dispatch this update to subscribers subscribed to both canonical or alternate URLs.

- "data": the content of the new version of this topic.

- "target" (optional): target audience of this update. This key can be present several times. See section #Authorization for further information.

- "id" (optional): the topic's revision identifier, it will be used as the SSE's "id" property, if omited the hub MUST generate a valid globally unique id. It "MAY" be an UUID.

- "type" (optional): the SSE's "event" property (a specific event type)

- "retry" (optional): the SSE's "retry" property (the reconnection time)

In case of success, the HTTP response's body MUST be the "id" associated to this update (the one generated by the hub, if it has not been provided by the client) and a success HTTP status code MUST be returned. The publisher MUST be authorized to publish updates. See section #Authorization.

5. Authorization

To ensure that they are authorized, both publishers and subscribers must present a valid JWS [RFC7515] in compact serialization to the hub. This JWS SHOULD be short lived, especially if the subscriber is a web browser. A different key MAY be used to sign subscribers' and publishers' tokens.

Two mechanisms are defined to present the JWS to the hub:

- using an "Authorization" HTTP header

- using a cookie

If a publisher or the subscriber is not a web browser, it SHOULD use an "Authorization" HTTP header. This "Authorization" header MUST contain the string "Bearer" followed by the JWS. The hub will check that the JWS is conform to the rules defined later to ensure that the client is authorized to publish or subscribe to updates.

By the "EventSource" specification, web browsers can not set custom HTTP headers for such connections, and they can only be estabilished using the "GET" HTTP method. However, cookies are supported, and can be included even in crossdomain requests if the CORS credentials are set (https://html.spec.whatwg.org/multipage/server-sent-events.html#dom-eventsourceinit-withcredentials):

If a publisher or a subscriber is a web browser, it SHOULD send a cookie called "mercureAuthorization" containing the JWS when connecting to the hub.

When possible, to improve the overall security, the "mercureAuthorization" cookie SHOULD be set during the discovery. See section #Discovery. Consequently, if the cookie is set during the discovery, both the publisher and the hub have to share the same second level domain. The "Domain" attribute MAY be used to allow the publisher and the hub to use different subdomains.

The cookie SHOULD have the "Secure", "HttpOnly" and "SameSite" attributes set. Setting the cookie's "Path" attribute SHOULD also be set to the hub's URL. See section #Security-Considerations.

When using authorization mechanisms, the connection MUST use an encryption layer such as HTTPS.

If both an "Authorization" HTTP header and a cookie named "mercureAuthorization" are presented by the client, the cookie MUST be ignored. If a client tries to execute an operation it is not allowed to, a 403 HTTP status code SHOULD be returned.

5.1. Publishers

Publishers MUST be authorized to dispatch updates to the hub, and MUST prove that they are allowed to send updates.

To be allowed to publish an update, the JWT presented by the publisher MUST contain a claim called "mercure", and this claim MUST contain a "publish" key. JWT's "mercure.publish" contains an array of targets the publisher is allowed to dispatch updates to.

If "mercure.publish":

- is not defined, then the publisher MUST NOT be authorized to dispatch any update

- contains an empty array, then the publisher is only allowed to dispatch public updates

- contains the reserved string "*" as an array value, then the publisher is authorized to dispatch updates to all targets

If a topic is not public, the "POST" request sent by the publisher to the hub MUST contain a list of keys named "target". Theirs values MUST be of type "string", and it is RECOMMENDED to use valid IRIs. They can be, for instance a user ID, or a list of group IDs. If an update contains at least one target the publisher is not authorized for, the hub MUST NOT dispatch the update (even if some targets in the list are allowed) and SHOULD return a 403 HTTP status code.

5.2. Subscribers

Subscribers MAY need to be authorized to connect to the hub. To receive updates destined to specific targets, they MUST be authorized, and MUST prove they belong to at least one of the specified targets. If the subscriber is not authorized it MUST NOT receive any update having at least one target.

To receive updates destined to specific targets, the JWS presented by the subscriber MUST have a claim named "mercure" with a key named "subscribe" that contains an array of strings: the list of targets the user is authorized to receive updates for. The targets SHOULD be IRIs.

If at least one target is specified, the update MUST NOT be sent to the subscriber by the hub, unless the "mercure.subscribe" property of the JWS presented by the subscriber contains at least one of the specified targets.

If the "mercure.subscribe" array contains the reserved string value "*", then the subscriber is authorized to receive updates destined to all targets.

6. Re-Connection and State Reconciliation

To allow re-establisment in case of connection lost, events dispatched by the hub SHOULD include an "id" property. The value contained in this "id" property SHOULD be a globally unique identifier. To do so, UUID [RFC4122] MAY be used.

According to the server-sent events specification, in case of connection lost the subscriber will try to automatically reconnect. During the reconnection the subscriber MUST send the last received event id in a Last-Event-ID (https://html.spec.whatwg.org/multipage/iana.html#last-event-id) HTTP header.

The server-sent events specification doesn't allow to set this HTTP header during the first connection (before a re-connection occurs). In order to fetch any update dispatched between the initial resource generation by the publisher and the connection to he hub, the subscriber MUST send the event id provided during the discovery in the "last-event-id" link's attribute in a query parameter named "Last-Event-ID" when connecting to the hub.

If both the "Last-Event-ID" HTTP header and the query parameter are present, the HTTP header MUST take precedence.

If the "Last-Event-ID" header or query parameter exists, the hub SHOULD send to the subscriber all events published since the one having this identifier.

The hub MAY discard some messages for operational reasons. The subscriber MUST NOT assume that no update will be lost, and MUST re-fetch the original topic to ensure this (for instance, after a long deconnection time).

The hub MAY also specify the reconnection time using the "retry" key, as specified in the server-sent events format.

7. Encryption

Using HTTPS doesn't prevent the hub to access to the update's content. Depending of the intended privacy of informations contained in the updates, it MAY be necessary to prevent eavesdropping by the hub.

To make sure that the message content can not be read by the hub, the publisher MAY encode the message before sending it to the hub. The publisher SHOULD use JSON Web Encryption [RFC7516] to encrypt the update content. The publisher MAY provide the relevant encryption key(s) in the "key-set" attribute of the Link HTTP header during the discovery. The "key-set" attribute SHOULD contain a key encoded using the JSON Web Key Set [RFC7517] format. Any other out-of-band mechanism MAY be used instead to share the key between the publisher and the subscriber.

Updates encyption is considered a best practice to prevent mass surveillance. This is especially relevant if the hub is managed by an external provider.

8. Security Considerations

The confidentiality of the secret key(s) used to generate the JWTs is a primary concern. The secret key(s) MUST be stored securely. They MUST be revoked immediatly in case of compromission.

Possessing a valid JWTs allows any client to subscribe, or to publish to the hub. Their confidentiality MUST therefore be ensured. To do so, JWTs MUST only be transmited over secure connections.

Also, when the client is a web browser, to be resilient to Cross-site Scription (XSS) attacks (https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)), the JWT SHOULD not be made accessible to JavaScript scripts. It's main reason why, when the client is a web browser, using "HttpOnly" cookies as authorization mechanism SHOULD always be prefered.

In case of compromission, revoking a JWT before its expiration is often difficult. So, using short-lived token is strongly RECOMMENDED.

The publish endpoint of the hub may be targeted by Cross-Site Request Forgery (CSRF) attacks (https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)) when the cookie-based authorization mechanism is used. Therefore, implementations supporting this mechanism MUST mitigate such attacks.

The first prevention method to implement is to set the "mercureAuthorization" cookie's "SameSite" attribute. However, some web browsers still not support this attribute (https://caniuse.com/#feat=same-site-cookie-attribute), and will stay vulnerable. In addition, hub implementations SHOULD use the "Origin" and "Referer" HTTP headers set by web browsers to verify that the source origin matches the target origin. If none of these headers are available, the hub SHOULD discard the request.

CSRF prevention techniques, including the ones previously mentioned, are described in depth in OWASP's Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet (https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet).

9. References

9.1. Normative References

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

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

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

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

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

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

9.2. Informative References

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <https://www.rfc-editor.org/info/rfc4287>.

[RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <https://www.rfc-editor.org/info/rfc6902>.

[RFC7386] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7386, DOI 10.17487/RFC7386, October 2014, <https://www.rfc-editor.org/info/rfc7386>.

[W3C.REC-html52-20171214] Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, 14 December 2017, <https://www.w3.org/TR/2017/REC-html52-20171214>.

[W3C.REC-json-ld-20140116] Sporny, M., Kellogg, G., and M. Lanthaler, "JSON-LD 1.0", World Wide Web Consortium Recommendation REC-json-ld-20140116, 16 January 2014, <http://www.w3.org/TR/2014/REC-json-ld-20140116>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, 26 November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

Author's Address

Kevin Dunglas
Les-Tilleuls.coop
5 rue Hegel
Lille 59000
France

Email: kevin@les-tilleuls.coop