XEP-0235: OAuth Over XMPP

Peter Saint-Andre
mailto:stpeter@stpeter.im
xmpp:stpeter@jabber.org
https://stpeter.im/

2009-03-24
Version 0.7

Status	Type	Short Name
Deferred	Standards Track	NOT_YET_ASSIGNED

This specification defines an XMPP extension for delegating access to protected resources over XMPP, using the OAuth protocol.

Legal

Copyright

This XMPP Extension Protocol is copyright © 1999 – 2024 by the XMPP Standards Foundation (XSF).

Permissions

Permission is hereby granted, free of charge, to any person obtaining a copy of this specification (the "Specification"), to make use of the Specification without restriction, including without limitation the rights to implement the Specification in a software program, deploy the Specification in a network service, and copy, modify, merge, publish, translate, distribute, sublicense, or sell copies of the Specification, and to permit persons to whom the Specification is furnished to do so, subject to the condition that the foregoing copyright notice and this permission notice shall be included in all copies or substantial portions of the Specification. Unless separate permission is granted, modified works that are redistributed shall not contain misleading information regarding the authors, title, number, or publisher of the Specification, and shall not claim endorsement of the modified works by the authors, any organization or project to which the authors belong, or the XMPP Standards Foundation.

Warranty

NOTE WELL: This Specification is provided on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE.

Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall the XMPP Standards Foundation or any author of this Specification be liable for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising from, out of, or in connection with the Specification or the implementation, deployment, or other use of the Specification (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if the XMPP Standards Foundation or such author has been advised of the possibility of such damages.

Conformance

This XMPP Extension Protocol has been contributed in full conformance with the XSF's Intellectual Property Rights Policy (a copy of which can be found at <https://xmpp.org/about/xsf/ipr-policy> or obtained by writing to XMPP Standards Foundation, P.O. Box 787, Parker, CO 80134 USA).

Contents

1 Introduction 1

2 Protocol Flow 1

3 Access Request Format 3

4 Signature Generation Algorithm 3

5 Error Handling 4

6 Determining Support 5

7 Security Considerations 6
- 7.1 Replay Attacks . . . 6
- 7.2 Encryption . . . 6

8 IANA Considerations 7

9 XMPP Registrar Considerations 7
- 9.1 Protocol Namespaces . . . 7
- 9.2 Protocol Versioning . . . 7

10 XML Schema 7
- 10.1 Protocol Namespace . . . 7
- 10.2 Error Namespace . . . 8

11 Acknowledgements 9

1 Introduction

Although authentication is required in order to access the XMPP network, in some situations it is desirable to require authorization in order for an authenticated entity to access certain resources on the network. For example, authorization may be required to join a Multi-User Chat (XEP-0045) [1] room, subscribe to a Publish-Subscribe (XEP-0060) [2] node, or to access other resources of interest (such as a media relay or communications gateway).
Dedicated technologies exist for authorization. One such technology is OAuth [3], as defined at <http://oauth.net/core/1.0/>. In the language of OAuth, a User can authorize a Consumer to access a Protected Resource that is hosted by a Service Provider; this authorization is encapsulated in a token that the User requests from the Service Provider, that the User shares with the Consumer, and that the Consumer then presents to the Service Provider in an access request.
This specification assumes that OAuth Access Tokens will be acquired outside the XMPP (i.e., via HTTP as defined in the core OAuth specification) and merely presented over XMPP when sending a protocol-specific access request.

2 Protocol Flow

The typical scenario is for a Consumer to request the authorization to act as a delegated authority on behalf of the User to access a Protected Resource owned by the User at a Service Provider. For example, the owner of a pubsub node could allow a remote entity to publish to that node (the single lines "---" show protocol flows over HTTP and the double lines "===" show protocol flows over XMPP):

```
Consumer                    Service Provider
   |                                |
   |  request a Request Token       |
   |----------------------------->|
   |  grant the Request Token       |
   |<-----------------------------|
   |                                |
   |  [Consumer redirects User      |
   |   to Service Provider]         |
   |                                |
   |  request an Access Token       |
   |----------------------------->|
   |  grant the Access Token        |
   |<-----------------------------|
   |  access Protected Resource     |
   |============================>|
```

[1] XEP-0045: Multi-User Chat <https://xmpp.org/extensions/xep-0045.html>.
[2] XEP-0060: Publish-Subscribe <https://xmpp.org/extensions/xep-0060.html>.
[3] OAuth Core 1.0 <http://oauth.net/core/1.0/>.

Before presenting an access token to a Service Provider in a protocol-specific access request, a Consumer SHOULD verify that the Service Provider supports this protocol, as described under the Determining Support section of this document.
Consider the example of a User (say, <world-traveler@example.com>) who wishes to authorize a Consumer (say, an application called FindMeNow as represented by the JID <travelbot@findmenow.tld>) to access the User's geolocation feed at a Service Provider called WorldGPS (as represented by a publish-subscribe node of <feeds.worldgps.tld/world-traveler>). The order of events might be as follows.

1. FindMeNow has registered as a Consumer for WorldGPS' API and has been assigned an OAuth consumer key and secret for use in its dealings with WorldGPS.
2. The User registers with WorldGPS, which creates a feed for the User's location data in an XMPP PubSub Node at WorldGPS.
3. The User visits FindMeNow.tld and requests real-time updates from his WorldGPS feed.
4. FindMeNow, over HTTP, requests an OAuth "request token" from WorldGPS, signing it with FindMeNow's OAuth consumer key and secret.
5. WorldGPS, if the signature was valid, sends FindMeNow an OAuth "request token."
6. FindMeNow then redirects the user to a WorldGPS webpage.
7. On the WorldGPS webpage, the User logs in (or is already logged in) and is then asked whether to approve of FindMeNow having read-only access to his geolocation information.
8. The User approves the request and WorldGPS redirects the User back to FindMeNow.
9. FindMeNow, over HTTP, requests an OAuth "access token" from WorldGPS, signing the request with the "request token" that has now been approved by the User.
10. WorldGPS, if the signature is correct and the request token was approved, replies to FindMeNow with an OAuth "access token".
11. FindMeNow, over XMPP, subscribes to the User's pubsub node using the OAuth "access token" as described below.

As a result, FindMeNow gets updated every time the User publishes items to his geolocation node at WorldGPS.
Steps 1-10 describe OAuth's standard HTTP flow and represent an out-band means for obtaining OAuth access tokens for use in XMPP operations.

3 Access Request Format

The access request MUST include the following parameters:

- oauth_consumer_key
- oauth_nonce
- oauth_signature (note: in XMPP this value is **not** escaped)
- oauth_signature_method
- oauth_timestamp
- oauth_token

The access MAY also include the "oauth_version" parameter
An example follows.

Listing 1: Pubsub subscription request with OAuth access token

```
<iq from='travelbot@findmenow.tld/bot'
    id='sub1'
    to='feeds.worldgps.tld'
    type='set'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe jid='travelbot@findmenow.tld' node='bard_geoloc'/>
    <oauth xmlns='urn:xmpp:oauth:0'>
      <oauth_consumer_key>0685bd9184jfhq22</oauth_consumer_key>
      <oauth_nonce>4572616e48616d6d65724c61686176</oauth_nonce>
      <oauth_signature>9PQkM4YKgaM067wqrDGshXOwDW0=</oauth_signature>
      <oauth_signature_method>HMAC-SHA1</oauth_signature_method>
      <oauth_timestamp>1218137833</oauth_timestamp>
      <oauth_token>ad180jjd733klru7</oauth_token>
      <oauth_version>1.0</oauth_version>
    </oauth>
  </pubsub>
</iq>
```

4 Signature Generation Algorithm

When sending an OAuth access request over XMPP, the signature method SHOULD be HMAC-SHA1. The Signature Base String SHALL be constructed from the following items:

- The HTTP request method SHALL be the qname of the XMPP stanza element used, that is, either "message" or "presence" or "iq".

- The request URL SHALL be the 'from' address of the XMPP stanza concatenated with the ampersand character "&" and the 'to' address of the XMPP stanza.
- The normalized request parameters string SHALL be all of the oauth_* parameters included in the <oauth/> element (except oauth_signature).

As an example, consider the stanza shown above.
The Signature Base String would be as follows (where line endings have been added for readability and are denoted by the "\" character):

```
iq%26travelbot%40findmenow.tld%2Fbot%26feeds.worldgps.tld%26\
oauth_consumer_key%3D0685bd9184jfhq22%26\
oauth_nonce%3D4572616e48616d6d65724c61686176%26\
oauth_signature_method%3DHMAC-SHA1%26\
oauth_timestamp%3D1218137833%26\
oauth_token%3Dad180jjd733klru7%26\
oauth_version%3D1.0
```

Assuming a consumer secret of 'consumersecret' and a token secret of 'tokensecret', the signature will be:

```
9PQkM4YKgaM067wqrDGshXOwDW0=
```

5 Error Handling

If a Service Provider rejects a Consumer's request to access a Protected Resource over XMPP, the Service Provider MUST return an XMPP stanza error. The XMPP error condition SHOULD be either <bad-request/> or <not-authorized/> and the stanza SHOULD include an OAuth-specific error condition as described in the following table.

OAuth-Specific Condition	Generic Condition	Description
<duplicated-parameter/>	<bad-request/>	One of the oauth_* elements was included more than once.
<invalid-consumer-key/>	<not-authorized/>	The Consumer's OAuth consumer key is not valid.
<invalid-nonce/>	<not-authorized/>	The provided nonce is invalid; it might have already been used.
<invalid-signature/>	<not-authorized/>	The provided signature is invalid; the Consumer needs to confirm that the signature base string is calculated correctly.

OAuth-Specific Condition	Generic Condition	Description
<invalid-token/>	<not-authorized/>	The provided access token is invalid; it might have been revoked.
<missing-parameter/>	<bad-request/>	One of the required oauth_* elements is missing.
<token-required/>	<not-authorized/>	The Consumer did not include an OAuth access token in its request; this error condition is XMPP-specific and does not have a counterpart in the OAuth specification.
<unsupported-parameter/>	<bad-request/>	The <oauth/> stanza contains unknown or unsupported parameters.
<unsupported-signature-method/>	<bad-request/>	The specified signature method is not supported by the server.

An example follows.

Listing 2: OAuth-specific error

```
<iq from='feeds.worldgps.tld'
    id='sub1'
    to='travelbot@findmenow.tld/bot'
    type='error'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <invalid-nonce xmlns='urn:xmpp:oauth:0:errors'/>
  </error>
</iq>
```

6 Determining Support

If an entity supports the protocol specified herein, it MUST advertise that fact by returning a feature of "urn:xmpp:oauth:0" in response to Service Discovery (XEP-0030) [4] information requests (see Protocol Namespaces regarding issuance of one or more permanent namespaces).

[4] XEP-0030: Service Discovery <https://xmpp.org/extensions/xep-0030.html>.

Listing 3: Service discovery information request

```
<iq from='travelbot@findmenow.tld/bot'
    id='disco1'
    to='feeds.worldgps.tld'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>
```

Listing 4: Service discovery information response

```
<iq from='feeds.worldgps.tld'
    id='disco1'
    to='travelbot@findmenow.tld/bot'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='urn:xmpp:oauth:0'/>
  </query>
</iq>
```

In order for an application to determine whether an entity supports this protocol, where possible it SHOULD use the dynamic, presence-based profile of service discovery defined in Entity Capabilities (XEP-0115) [5]. However, if an application has not received entity capabilities information from an entity, it SHOULD use explicit service discovery instead.

7 Security Considerations

7.1 Replay Attacks

Signatures generated according to the signature generation algorithm might be subject to replay attacks. However, inclusion of the XMPP "from" and "to" addresses limits these attacks to compromised servers or client-to-server connections. In addition, inclusion of the nonce value also helps to prevent replay attacks.

7.2 Encryption

OAuth tokens SHOULD be sent only over TLS-encrypted client-to-server connections, and all server-to-server connections SHOULD be TLS-enabled. Additional security can be provided using appropriate methods for the end-to-end encryption of XMPP traffic, such as Current Jabber OpenPGP Usage (XEP-0027) [6], RFC 3923 [7] Encrypted Session Negotiation (XEP-0116) [8],

[5] XEP-0115: Entity Capabilities <https://xmpp.org/extensions/xep-0115.html>.
[6] XEP-0027: Current Jabber OpenPGP Usage <https://xmpp.org/extensions/xep-0027.html>.
[7] RFC 3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP) <http://tools.ietf.org/html/rfc3923>.
[8] XEP-0116: Encrypted Session Negotiation <https://xmpp.org/extensions/xep-0116.html>.

or End-to-End XML Streams (XEP-0246) [9].

8 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [10].

9 XMPP Registrar Considerations

9.1 Protocol Namespaces

This specification defines the following XML namespace:

- urn:xmpp:oauth:0

Upon advancement of this specification from a status of Experimental to a status of Draft, the XMPP Registrar [11] shall add the foregoing namespaces to the registry located at <https://xmpp.org/registrar/namespaces.html>, as described in Section 4 of XMPP Registrar Function (XEP-0053) [12].

9.2 Protocol Versioning

If the protocol defined in this specification undergoes a revision that is not fully backwards-compatible with an older version, the XMPP Registrar shall increment the protocol version number found at the end of the XML namespaces defined herein, as described in Section 4 of XEP-0053.

10 XML Schema

10.1 Protocol Namespace

[9] XEP-0246: End-to-End XML Streams <https://xmpp.org/extensions/xep-0246.html>.

[10] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[11] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.

[12] XEP-0053: XMPP Registrar Function <https://xmpp.org/extensions/xep-0053.html>.

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:xmpp:oauth:0'
    xmlns='urn:xmpp:oauth:0'
    elementFormDefault='qualified'>

  <xs:element name='oauth'>
    <xs:complexType>
      <xs:choice>
        <xs:element name='oauth_consumer_key' type='xs:string'/>
        <xs:element name='oauth_nonce' type='xs:string'/>
        <xs:element name='oauth_signature' type='xs:string'/>
        <xs:element name='oauth_signature_method' type='xs:string'/>
        <xs:element name='oauth_timestamp' type='xs:string'/>
        <xs:element name='oauth_token' type='xs:string'/>
        <xs:element name='oauth_version' type='xs:string'/>
      </xs:choice>
    </xs:complexType>
  </xs:element>

</xs:schema>
```

10.2 Error Namespace

```
<?xml version='1.0' encoding='UTF-8'?>

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='urn:xmpp:oauth:0:errors'
    xmlns='urn:xmpp:oauth:0:errors'
    elementFormDefault='qualified'>

  <xs:element name='duplicated-parameter' type='empty'/>
  <xs:element name='invalid-consumer-key' type='empty'/>
  <xs:element name='invalid-nonce' type='empty'/>
  <xs:element name='invalid-signature' type='empty'/>
  <xs:element name='invalid-token' type='empty'/>
  <xs:element name='missing-parameter' type='empty'/>
  <xs:element name='token-required' type='empty'/>
  <xs:element name='unsupported-parameter' type='empty'/>
  <xs:element name='unsupported-signature-method' type='empty'/>

  <xs:simpleType name='empty'>
    <xs:restriction base='xs:string'>
      <xs:enumeration value=''/>
```

```
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```

11 Acknowledgements

The author gratefully acknowledges the contributions of Blaine Cook, Leah Culver, Kellan Elliott-McCrea, Seth Fitzsimmons, Nathan Fritz, Evan Henshaw-Plath, Joe Hildebrand, and Ralph Meijer to the content of this specification, as provided during the XMPP Summit held in Portland, Oregon, on July 21 and 22, 2008. Thanks also to Dave Cridland and Pedro Melo for their comments on an early draft. Seth Fitzsimmons checked many details and provided text regarding the protocol flow and error handling.