XEP-0493: OAuth Client Login

Matthew Wild
mailto:mwild1@gmail.com
xmpp:me@matthewwild.co.uk

2025-03-31
Version 0.1.1

Status	Type	Short Name
Experimental	Informational	tsa

This specification details how a third-party can be securely granted access to an XMPP account without exposing the account credentials, using OAuth.

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**
- 1.1 Use of OAuth . . . 1
- 1.2 Defining the scope of this document . . . 1

2 Requirements **2**
- 2.1 Use cases . . . 2
- 2.2 Client login flow . . . 3
- 2.3 Obtaining authorization . . . 3
  - 2.3.1 Discovering OAuth support . . . 3
  - 2.3.2 Registering with the OAuth provider . . . 4
  - 2.3.3 Authorization request . . . 4
  - 2.3.4 Successful authorization . . . 4

3 OAuth Scopes **5**
- 3.1 Special scopes . . . 5
  - 3.1.1 Account-only access . . . 5

4 Implementation notes **6**
- 4.1 OAuth versions . . . 6
- 4.2 Servers . . . 6

5 Security considerations **7**
- 5.1 OAuth security . . . 7
- 5.2 Authentication vs Authorization . . . 7

6 Accessibility Considerations **8**

7 Privacy Considerations **8**

8 IANA Considerations **8**

9 XMPP Registrar Considerations **9**
- 9.1 Scope name registry . . . 9
  - 9.1.1 Initial contents . . . 9

10 Acknowledgements **9**

1 Introduction

This specification defines a way to allow an XMPP account owner to safely and securely log in to their account through a client or other application without sharing their password or other primary credentials with that application. This is particularly important if the application is hosted by a third party (e.g. as a web app), as some XMPP applications are.
To achieve this, we define how OAuth can be implemented and used to grant this access.

1.1 Use of OAuth

OAuth is a widely used authentication and authorization framework. It allows a resource owner (such as an XMPP user) to safely and securely grant address to a resource (such as their XMPP account) to software and services that they choose. In the past, granting access to an account was only possible by sharing the account password with the software or service that wants to access it. Once shared, such access is unlimited and impossible to selectively revoke. However, OAuth is also a very broad framework with many features and capabilities, spread across multiple specifications. This can make it difficult for developers to know which parts are necessary for different usage scenarios, and this leads to non-interoperable implementations, or avoiding implementation at all.

1.2 Defining the scope of this document

This specification focuses on one primary use case - securely granting third-party access to an account. It does not introduce any new protocols, but describes how existing protocols can be combined together to enable this use case.
There is an existing published XEP related to use of OAuth in XMPP, OAuth Over XMPP (XEP-0235) [1]. However this describes a different use case - specifically, how an arbitrary XMPP entity can use OAuth to manage access to its own services by other XMPP entities. Furthermore, that specification is based on the outdated OAuth 1.0, while this specification builds upon OAuth 2.0+ only.
Other use cases that are **not** described by this document, but may also be achieved using OAuth include:

- Allowing users to identify themselves to a third-party service by proving ownership of an XMPP account (e.g. this would allow a web service to display a "Log in with XMPP" button). Such a feature could be implemented by adding support for OpenID Connect protocols in addition to the simpler protocols described here. Alternatively, Verifying HTTP Requests via XMPP (XEP-0070) [2] defines a pre-OAuth method of achieving similar functionality.

[1] XEP-0235: OAuth Over XMPP <https://xmpp.org/extensions/xep-0235.html>.
[2] XEP-0070: Verifying HTTP Requests via XMPP <https://xmpp.org/extensions/xep-0070.html>.

- Allowing users to identify themselves to their XMPP server using a third-party service (e.g. this would allow an XMPP client to display a "Log in with TrendyService" button). This can be achieved through use of OpenID Connect, which has significant overlap with the parts of OAuth used in this specification, but ultimately requires different protocol flows.

By limiting the scope of this document to secure account access delegation, we aim to make it easier for developers to safely and securely implement this important feature, without the distractions of the complex and sprawling wider OAuth ecosystem.
However, this XEP's protocols do unlock additional use cases. For example, the flows described here would enable server admins to deploy and utilize OAuth-compatible identity providers to manage access to XMPP user accounts, which in turn can be used for features such as Single Sign-On.

2 Requirements

- It should be possible for an XMPP account owner to provide an application or service access to their account without sharing their password (or any similar unconstrained credentials).

Although the protocol details will not be covered in this particular document, following this specification must enable the following important features to be implemented by the service:

- It should be possible to control the level of the access granted (e.g. providing read-only access, or allowing access to only certain aspects of the account).
- It should be possible to revoke the granted access at any time.

2.1 Use cases

Imagine that a developer has made an XMPP web application that allows people to create and browse XMPP social content (such as Microblogging Over XMPP (XEP-0277) [3]) on the XMPP network. To work, it needs to connect to a user's XMPP account. However the developer does not want the liability of requesting (and potentially storing) user passwords to allow the application to log in to user accounts. Additionally, users do not want to type their passwords into a third-party service.
Using OAuth, the user is able to enter only their JID in the application, and then through OAuth negotiation the application will be able to obtain unique credentials that can be used to connect to the user's XMPP account.

[3] XEP-0277: Microblogging over XMPP <https://xmpp.org/extensions/xep-0277.html>.

2.2 Client login flow

The initial login process can be broken down into a series of steps, which are each given a section here, in the order they will typically occur.
The following entities play a part in these sections:

- The ”user” - this is the, typically human, individual that wants to use the application with their XMPP account.
- The ”application” (or in the context of XMPP, also referred to as the ”client”) - this is the application or service that wants to access the user’s account on behalf of the user.
- The ”server” - this is the user’s XMPP service, identified by the domain part of their JID.
- The ”OAuth provider” - this is the service with which the application performs OAuth. It may be part of the user’s XMPP server, but may also be located elsewhere. The OAuth provider is responsible for issuing credentials that can be used to access the user’s account.

2.3 Obtaining authorization

To begin the process, the application must first obtain the user’s JID. For example, it may present a login form with a field for the JID (but no field for the password).

2.3.1 Discovering OAuth support

The application takes the hostname from the JID, and initiates a client connection to the user’s account on that server. Per XMPP Core [4], the client SHOULD include the user’s JID in the ’from’ attribute on opening stream headers, but MUST only do so after the connection is secured with TLS.
The server will present a list of SASL mechanisms, either using the core SASL profile defined in XMPP Core [5], or using Extensible SASL Profile (XEP-0388) [6].
If the server does not offer the OAUTHBEARER mechanism in this list, then it does not support the flow defined in this XEP. The application SHOULD indicate to the user that their server does not support secure third-party application access and abort the login process. However, the application MAY offer password login as a fallback, if such fallback has been enabled by the administrator of the application deployment.
During initial authentication, the client does not have any credentials, so it SHOULD proceed with the OAUTHBEARER mechanism, but provide an empty access token. Per the rules in RFC 7628 [7] , the server will respond with a JSON object that includes the discovery URL for the

[4] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[5] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[6] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.
[7] RFC 7628: A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth <http://tools.ietf.org/html/rfc7628>.

requested user.
The application makes a request to the discovery URL to fetch metadata about the OAuth provider, which is needed to complete the authorization process. The response from the discovery URL SHOULD be parsed according to RFC 8414 [8] .

2.3.2 Registering with the OAuth provider

If the application has not previously interacted with this OAuth provider (identified by the "issuer" URL in the metadata), then it must first register itself. This registration is necessary to protect against certain attacks on the OAuth process, and it also provides an opportunity for the application to provide details such as a name and logo that will be presented to the user. Details of this registration process are specified in RFC 7591 [9] .

2.3.3 Authorization request

Authorization begins by preparing an authorization request. In the case of server-side "hosted" applications, this request is not made directly by the application. Rather, the application must craft the URL (known as the "Authorization Endpoint") for the request and then redirect the user to that URL.
The base of the Authorization Endpoint is obtained from the earlier discovery process. The client then adds parameters specific to the current authorization request. XMPP implementations SHOULD implement at least the Authorization Code grant type (RFC 6749 [10] section 4.1), and details of the expected parameters for this grant type can be found in RFC 6749 [11] section 4.1.1. In addition, servers and clients MUST support the PKCE extension to this flow, which is defined in RFC 7636 [12] .
The application SHOULD request specific scopes, according to the access that it requires. These scopes MUST be limited to the minimum level of access required for the application to function. OAuth itself does not define any standard scopes, however some standard scopes are included in this specification that are expected to be offered, in addition to any custom scopes that a specific service or implementation may offer.
At this point, the OAuth provider will display to the user whatever steps are necessary to grant the authorization, such as requiring the user to authenticate themselves.

2.3.4 Successful authorization

If the user successfully approves the application's access, the OAuth provider will redirect them back to the application (specifically, the application's "Redirection URI" which it

[8] RFC 8414: OAuth 2.0 Authorization Server Metadata <http://tools.ietf.org/html/rfc8414>.
[9] RFC 7591: OAuth 2.0 Dynamic Client Registration Protocol <http://tools.ietf.org/html/rfc7591>.
[10] RFC 6749: The OAuth 2.0 Authorization Framework <http://tools.ietf.org/html/rfc6749>.
[11] RFC 6749: The OAuth 2.0 Authorization Framework <http://tools.ietf.org/html/rfc6749>.
[12] RFC 7636: Proof Key for Code Exchange by OAuth Public Clients <http://tools.ietf.org/html/rfc7636>.

provided to the server previously in the Authorization Request or Client Registration).
In the case of the Authorization Code grant type, the application will be able to exchange the authorization code received in the URL for the final credentials - an access token that can be used with the SASL OAUTHBEARER mechanism. This exchange is described in RFC 6749 [13] sections 4.1.3 and 4.1.4.

3 OAuth Scopes

OAuth allows each authorization grant to have different permissions, it calls these ”scopes”. However, because OAuth is a generic framework, it does not specify any scopes itself. This can become an interoperability issue, if applications don't understand what scopes they need to request for the functionality they implement.
For this reason, we define some basic scopes in this specification, without excluding the possibility that additional scopes will be defined in the future.
Scope names SHOULD be registered with the XSF before being used, however this is not a requirement. Custom scopes that are not intended for standardization SHOULD avoid the 'xmpp:' prefix to prevent conflicts. Note that scope names are arbitrary opaque strings as far as OAuth is concerned, and are not defined to be URIs, or any specific format.
This specification defines the following scopes:

- xmpp:client:normal - The application gains access to most features of the account, but may be restricted from certain sensitive operations, such as changing the account password or managing connected devices. This ensures that such a client cannot escalate its privileges or take over an account. This scope MUST be supported by the server.

3.1 Special scopes

For certain specialized non-IM applications, it may be preferable to request limited access to a user's account. For example, an application which allows a user to view or export or backup their data.
These scopes do not permit communication access to the XMPP network. They SHOULD be supported by servers.

3.1.1 Account-only access

These scopes control access to account data (including user profile information, public and private PEP nodes, and roster). In general they allow communication with the account's server to facilitate this access, but not with other XMPP entities. Notably they do not grant access to a user's current or past communications (e.g. Message Carbons (XEP-0280) [14] or

[13] RFC 6749: The OAuth 2.0 Authorization Framework <http://tools.ietf.org/html/rfc6749>.
[14] XEP-0280: Message Carbons <https://xmpp.org/extensions/xep-0280.html>.

Message Archive Management (XEP-0313) [15]).

- xmpp:account:read - Read-only access to the account and associated data.
- xmpp:account:write - Write access to the account and associated data.

4 Implementation notes

4.1 OAuth versions

At the time of writing, OAuth 2.0 is published as RFCs and widely supported by implementations and libraries generally. OAuth 2.1 is under development, and while broadly compatible it has many improvements that improve robustness, increase interoperability and reduce the potential for security issues.
The subset of OAuth required by this specification is not likely to change substantially, and it is probable that a future version of this XEP may advise support for OAuth 2.1 when it is finalized. Implementers are encouraged to cross-reference the OAuth 2.1 drafts, as they can be simpler than the original OAuth 2.0 specifications while the protocol remains largely identical.
In particular, OAuth 2.1 requires the PKCE extension to the Authorization Grant flow, which this XEP already requires. Many of the other OAuth 2.0 grant types are removed from OAuth 2.1, and this XEP does not recommend that implementations support these grant types.
Finally, OAuth 2.1 formally forbids the practice of allowing wildcards in the client's redirect_uri fields. Allowing fuzzy matching can lead to unintented security issues, and a simple string comparison should be used instead, aside from special handling of certain localhost URLs (see the OAuth 2.1 specification for details).
Following the few paragraphs above will lead to implementations which are simpler, safer and forwards-compatible with OAuth 2.1.

4.2 Servers

At a minimum, servers are required to implement the OAUTHBEARER SASL mechanism defined in RFC 7628 [16] . To ensure clients can adequately discover support and the information they need to successfully obtain authorization, servers MUST provide a URL to a valid OAuth Discovery document (RFC 8414 [17]) in the openid-configuration field described in that RFC.
The discovery document, as well as the rest of the OAuth exchange, may be implemented and served by the XMPP server itself, or by an external OAuth-compatible identity provider that the server admin configures. This choice is left to individual implementations and

[15] XEP-0313: Message Archive Management <https://xmpp.org/extensions/xep-0313.html>.
[16] RFC 7628: A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth <http://tools.ietf.org/html/rfc7628>.
[17] RFC 8414: OAuth 2.0 Authorization Server Metadata <http://tools.ietf.org/html/rfc8414>.

deployments.

5 Security considerations

5.1 OAuth security

This specification builds upon the existing OAuth standards, this allows reuse of existing implementation experience and security practices.
Nevertheless, while OAuth has the potential to greatly improve security (compared to a world where users freely hand out their passwords to third-party services), there has been a long history of security issues related to OAuth implementations. Some general advice for developers:

- Read draft-ietf-oauth-security-topics [18] for an excellent review of known security pitfalls that can be encountered while implementing OAuth applications and services.
- Reuse existing libraries and APIs where possible, and read their documentation carefully.
- Do not implement or support OAuth 1.x, or any of the deprecated grant types.
- Read the security considerations in the RFCs this specification builds upon.

Server implementations MUST NOT support this specification without giving account owners a method to manage and revoke access. While this could be a proprietary interface, implementations SHOULD provide support for this using standard XMPP protocols, such as Client Access Management (XEP-0494)[19].
Be sure to review the 'OAuth versions' section of the Implementation Notes above and implement the requirements listed there for improved security and for compatibility with OAuth 2.1 when it arrives.

5.2 Authentication vs Authorization

Note well that this specification is about an XMPP account owner granting (i.e. authorizing) an application access to their account. It is not about the account owner asserting any particular identity to the application, nor is is it designed to assert the identity of the application towards the XMPP service.
One simple example of how this confusion could manifest, is if a user initially provides one JID (which they may not own) to an application, but authenticates to the server using a different JID during the OAuth process, potentially confusing the application about the user's identity.

[18] OAuth 2.0 Security Best Current Practice <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-security-topics>.
[19] XEP-0494: Client Access Management <https://xmpp.org/extensions/xep-0494.html>.

OAuth is an authorization (permission) protocol, not an authentication (identity) protocol. Applications can safely use the credentials returned from OAuth negotiation to connect and authenticate to the XMPP service as the authenticated user, but should be careful not to make the assumption that this somehow authenticates the user to the application itself. Such authentication is not the intention of OAuth.
In the case of an XMPP client that establishes a session, note that the full JID of the authenticated session is returned by the server during resource binding, which allows the application to know which JID was ultimately authenticated.
Alternatively, the OpenID Connect protocols build upon OAuth to allow an application to securely learn the identity of an authenticated user. However, as described in the Requirements section earlier, these use cases are beyond the scope of this specification.
In all cases, a shared application that maintains local state (i.e. outside of the user's XMPP server) MUST be extremely careful to avoid leaking such data between user accounts, although such care must be taken with or without OAuth.

6 Accessibility Considerations

This specification potentially adds new human interaction points, such as during the authentication and authorization process. These steps will be provided by the XMPP server or an identity provider configured by the server administrator. These interfaces should be implemented and chosen according to accessibility best practices. Deployments should consider the accessibility impact of features such as CAPTCHAs which may be presented during an authentication flow, and ensure there are accessible variants if they must be used.

7 Privacy Considerations

As noted in the Security Considerations, implementations MUST support viewing and revoking authorizations after they have been granted.

8 IANA Considerations

None.

9 XMPP Registrar Considerations

9.1 Scope name registry

This specification introduces a registry of scope names. The format of an entry should be in the following format:

```
<scope>
  <name>Name of the OAuth scope to be used in the protocol, typically
     begins with 'xmpp:' prefix</name>
  <doc>Associated specification</doc>
  <desc>Natural-language description of the scope and its purpose</
     desc>
</scope>
```

9.1.1 Initial contents

Upon advancement of this XEP, the registrar will create the registry with the following initial contents:

```
<scope>
  <name>xmpp:client:normal</name>
  <doc>XEP-xxxx</doc>
  <desc>Permit general access to an XMPP account, excluding security-
     relevant features</desc>
</scope>

<scope>
  <name>xmpp:account:read</name>
  <doc>XEP-xxxx</doc>
  <desc>Permit access to read account data</desc>
</scope>

<scope>
  <name>xmpp:account:write</name>
  <doc>XEP-xxxx</doc>
  <desc>Permit access to modify account data</desc>
</scope>
```

10 Acknowledgements

Thanks to Jonas Schäfer for assisting in the review of this document, and to Kim Alvefur for the (almost) tireless assistance with developing the prototype implemention and making sense of the of various OAuth specifications with me. Thanks to NLnet for providing the

funding that made it possible for me to do this work on XMPP authentication improvements.