Intro to OAuth

IETF 110

Aaron Parecki • March 2021
aaronpk.com
The OAuth 2.0 Authorization Framework

Abstract

The OAuth 2.0 authorization framework enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf. This specification replaces and obsoletes the OAuth 1.0 protocol described in RFC 5849.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in section 1 of RFC 5741.

Information about the current status of this document, any issues or errata, and how to provide feedback on it can be obtained at https://www.rfc-editor.org/rfc/rfc6749

Copyright Notice

Copyright (c) 2013 The IETF Trust and the persons identified as the Document Authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe the rights and restrictions with respect to this document. Code components extracted from this document must include Simplified BSD license text as described in Section 4.c of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

March Standards Track [Page 1]

RFC 6749

OAuth 2.0

October 2012

Table of Contents

1. Introduction ............................................. 4
  1.1. MO2O ........................................... 4
  1.2. Protocol flow ..................................... 4
  1.3. Authorization Grant .............................. 5
      1.3.1. Authorization Code ....................... 5
      1.3.2. Implicit ................................ 5
      1.3.3. Resource Owner Password Credentials .... 5
      1.3.4. Client credentials ....................... 5
      1.3.5. Access Token ................................ 4
      1.3.6. Refresh Token ............................ 4
      1.3.7. TLD Version ................................ 4

Specs are not good tutorials!
The Password Anti-Pattern

Are your friends already on Yelp?
Many of your friends may already be here, now you can find out. Just log in and we'll display all your contacts, and you can select which ones to invite! And don't worry, we don't keep your email password or your friends' addresses. We loathe spam, too.

Your Email Service
- msn Hotmail
- Yahoo! Mail
- AOL Mail
- Gmail

Your Email Address
ima.testguy@gmail.com (e.g. bob@gmail.com)

Your Gmail Password
(Password you use to log into your Gmail email)

Skip this step
Check Contacts
The Password Anti-Pattern

Step 1
Find Friends

Step 2
Profile Information

Step 3
Profile Picture

Are your friends already on Facebook?
Many of your friends may already be here. Searching your email account is the fastest way to find your friends on Facebook.

Gmail
Your Email: [input field]
Email Password: [input field]
Find Friends

* Facebook will not store your password.

Yahoo!
Find Friends

Windows Live Hotmail
Find Friends

Other Email Service
Find Friends
The Password Anti-Pattern

- How do you revoke this app’s access?
- Do you trust the app to not store your password?
- Do you trust the app to access only the things it says it needs?
- Do you trust the app to not do things like change your password or delete your account?
how can I let an app access my data without giving it my password?
OAuth doesn't tell the app who logged in.
Accessing APIs

- authorization

Identification

- authentication
How OAuth Works
Goal of the Client:

Get an access token

Use the access token to make API requests
OAuth Flows

Authorization Code

- web
- mobile
- SPA
- CLI

Device Flow

- CLI
- browserless devices

Client Credentials

- server-to-server
POST /resource/1/update HTTP/1.1
Authorization: Bearer RsT5OjbzRn430zqMLgV3Ia
Host: api.authorization-server.com
description=Hello+World
ROLES IN OAUTH

The User
(Resource Owner)

Device
(User Agent)

The Application
(Client)

OAuth Server
(Authorization Server)
aka the token factory

API
(Resource Server)
ROLES IN OAUTH

The User (Resource Owner)
Device (User Agent)
The Application (Client)
OAuth Server (Authorization Server) aka the token factory
Travis-CI.org
GitHub
API (Resource Server)
ROLES IN OAuth

- **The User** (Resource Owner)
- **Device** (User Agent)
- **The Application** (Client)
- **OAuth Server** (Authorization Server) aka the token factory
- **API** (Resource Server)

- iPhone App
- Okta
- Your API
Authorization Code + PKCE
Front Channel

Sent from client to server

HTTPS request from client to server, so requests cannot be tampered with

Back Channel

Passing data via the browser's address bar

The user, or malicious software, can modify the requests and responses

https://accounts.google.com/?...
Passing Data via the Back Channel
Passing Data via the Front Channel
User: I’d like to use this great app

App: Hang on while I generate a temporary secret and hash it

App: Please go to the authorization server to grant me access, take this hash with you

User: I’d like to log in to this app, here’s the hash it gave me

AS: Here is a temporary code the app can use

User: Here is the temporary code, please use this to get a token

App: Here's the code, and the temporary secret, please give me a token

AS: Let me verify the hash of that secret... ok here is an access token!

App: Please let me access this user’s data with this access token!
Ensures the app that receives the access token is the same one that started the exchange
Refresh tokens
Refresh tokens
keep the user logged in
Exchange the Refresh Token for an Access Token

POST https://authorization-server.com/token

grant_type=refresh_token&
refresh_token=REFRESH_TOKEN&
client_id=CLIENT_ID&
client_secret=CLIENT_SECRET
New Access Token in the Response

```
{
    "access_token": "RsT5Oj bzRn43OzqMLgV3Ia",
    "expires_in": 3600,
    "refresh_token": "64d049f8b21191e12522d5d96d5641af5e8"
}
```
authorization request

user authenticates

access token & refresh token

store refresh token in secure storage
already has refresh token

biometrics unlock refresh token

use refresh token to get new access token

new access token & refresh token
Scope
Scope lets an application request limited access to data
Quill will receive the following info: your friend list and email address.

Edit the info you provide

Privacy Policy
Example App by Aaron Parecki would like the ability to access the following data in your Fitbit account:

- Select All
- activity and exercise
- weight
- sleep
- food and water logs
- location and GPS
- profile
- heart rate

Deny | Allow

Data shared with aaronpk will be governed by Aaron Parecki’s privacy policy and terms of service. You can revoke this consent at any time in your Fitbit account settings. More information about these permissions can be found here.

Signed in as aaron@parecki.com
Not you?
The app requests certain scopes, and is confirmed by the user and the authorization server.
Access tokens
Access tokens are what the application uses to request data from the API.
Types of Access Tokens

Reference

MTQ0NjJkZmQ5OTM2NDE1ZTZjNGZmZjI3

Self-Encoded (e.g. JWT)

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEwMDAsImIiOj8vYXV0aG9yaXphdGlvbilzZXJ2ZXIuY29tIiwiy2lkIjoiaHR0cHM6Ly9leGFtcGxlLWFwcC5jb20iLCJpYXQiOjE0NzAwMDI3MDMsImV4cCI6MTUyOTE3NDg1MSwic2NvcGUiOiJyZWFkIHdyaXRlIn0.QiIrnmaC4VrbAYAsu0YPeuJ992p20fSxrXWPLw-gkFA
Reference Tokens

MTQ0NjJkZmQ5OTM2NDE1ZTZjNGZmZjI3

* user_id
* expiration
* permissions
* ...

Database?
Self-Encoded Tokens

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJEmMzMTI3NzA0NTM1OTQwODY1MjgyOTkxNTQwIiwib2x1dGlvbiI6WyJodHRwczovL2FwYWdlL3NpdGVzL2FwaS4iLCJlbnN1cmV0cmljIjoiY2xvY2F0aW9uIiwiaXNzIjoidW5kZWZpb24ifQ.WKg2Ft8MuaUcY5NM5XWIFzC4O65zPjorN8Rc0f1f3UA

{  
  "sub": "{USER_ID}",
  "aud": "{CLIENT_ID}",
  "exp": 1524240821,
  "scope": "create"
}
Access Token Validation

The Fast Way

Local Validation

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRtaW4iOnRydWUsImp0aSI6IjAyMDkzODIzNzAzMzYxNjIwIiwiaWF0IjoxMTg2MzUxNjQ4LCJleHAiOjE0MzY2NjIwMjYsImlhdCI6MTUzMDM2Mzk3OX0.eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRtaW4iOnRydWUsImp0aSI6IjAyMDkzODIzNzAzMzYxNjIwIiwiaWF0IjoxMTg2MzUxNjQ4LCJleHAiOjE0MzY2NjIwMjYsImlhdCI6MTUzMDM2Mzk3OX0

POST https://authorization-server.com/introspect
token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWRtaW4iOnRydWUsImp0aSI6IjAyMDkzODIzNzAzMzYxNjIwIiwiaWF0IjoxMTg2MzUxNjQ4LCJleHAiOjE0MzY2NjIwMjYsImlhdCI6MTUzMDM2Mzk3OX0

&client_id={CLIENT_ID}
&client_secret={CLIENT_SECRET}

The Strong Way

Remote Introspection

{...
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true,
  "jti": "b9d4a75b-0603-481c-842b4dd6",
  "iat": 1532400922,
  "exp": 1532404522
}
Rejecting Revoked Tokens

Remote Introspection
- Valid
- Valid
- Invalid
- Invalid
- Invalid
- Invalid
- Invalid
- Invalid

Local Validation
- Valid
- Valid
- Valid
- Valid
- Valid
- Valid
- Invalid
- Invalid

User revokes application
- Invalid
- Invalid
- Invalid
- Invalid
- Invalid
- Invalid
- Invalid
- Invalid
- Invalid

0:00 1:00 2:00 3:00 4:00 5:00 6:00 7:00 expired
Current Work
OAuth 2.1

Consolidate the OAuth 2.0 specs, adding best practices, removing deprecated features

Capture current best practices in OAuth 2.0 under a single name
OAuth 2.0

RFC6749 OAuth Core

Authorization Code
Implicit
Password
Client Credentials

RFC6750 Bearer Tokens

Tokens in HTTP Header
Tokens in POST Form Body
Tokens in GET Query String

PKCE for confidential clients
PKCE for SPAs
PKCE for mobile

Security BCP
OAuth 2.1

- Authorization Code + PKCE
- Client Credentials
- Tokens in HTTP Header
- Tokens in POST Form Body
OAuth 2.1

oauth.net/2.1

tools.ietf.org/html/draft-ietf-oauth-v2-1
JWT Profile for Access Tokens

Describes a standard set of JWT claims to use in a JWT access token.

This enables resource servers to be built with standard libraries to validate tokens.
Rich Authorization Requests (RAR)

oauth.net/2/rich-authorization-requests

{
    "type": "payment_initiation",
    "locations": [
        "https://example.com/payments"
    ],
    "instructedAmount": {
        "currency": "EUR",
        "amount": "123.50"
    },
    "creditorName": "Merchant123",
    "creditorAccount": {
        "iban": "DE02100100109307118603"
    },
    "remittanceInformationUnstructured": "Ref Number Merchant"
}
Pushed Authorization Requests (PAR)

oauth.net/2/pushed-authorization-requests

- Currently, the authorization request is sent in the front-channel
- Front-channel is susceptible to inspection and modification
- PAR initiates the OAuth flow from the back-channel
Specs Built on OAuth

- OpenID Connect (openid.net)
- FAPI (Financial-Grade API)
- UMA (User-Managed Access)
- IndieAuth (indieauth.net)