Network Working Group
Internet-Draft
Obsoletes: 5849 (if approved)
Intended status: Standards Track
Expires: July 25, 2011

E. Hammer-Lahav, Ed.
Yahoo!
D. Recordon
Facebook
D. Hardt
Microsoft

January 21, 2011

The OAuth 2.0 Authorization Protocol
draft-ietf-oauth-v2-12

Abstract

This specification describes the OAuth 2.0 authorization protocol.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 25, 2011.

Copyright Notice

Copyright (c) 2011 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 your 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.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
1.1. Roles 3
1.2. Access Token 5
1.3. Authorization Grant 6
1.4. Refresh Token 7
1.5. Notational Conventions 9
2. Client Authentication 9
2.1. Client Password Authentication 9
2.2. Other Client Authentication Methods 10
3. Protocol Endpoints 11
3.1. Authorization Endpoint 11
3.2. Token Endpoint 12
4. Requesting an Access Token 13
4.1. Authorization Code 13
4.2. Implicit Grant 19
4.3. Resource Owner Password Credentials 24
4.4. Client Credentials 26
4.5. Extensions 28
5. Issuing an Access Token 28
5.1. Successful Response 28
5.2. Error Response 30
6. Refreshing an Access Token 31
7. Accessing Protected Resources 32
7.1. Access Token Types 33
8. Extensibility 33
8.1. Defining Access Token Types 33
8.2. Defining New Endpoint Parameters 34
8.3. Defining New Authorization Grant Types 34
9. Security Considerations 34
10. IANA Considerations 35
10.1. The OAuth Access Token Type Registry 35
10.2. The OAuth Parameters Registry 36
Appendix A. Examples 39
Appendix B. Contributors 39
Appendix C. Acknowledgements 39
Appendix D. Document History 39
11. References 43
11.1. Normative References 43
11.2. Informative References 45
Authors' Addresses 45

1. Introduction

In the traditional client-server authentication model, the client accesses a protected resource on the server by authenticating with the server using the resource owner's credentials. In order to provide third-party applications access to protected resources, the resource owner shares its credentials with the third-party. This creates several problems and limitations:

- Third-party applications are required to store the resource-owner's credentials for future use, typically a password in clear-text.
- Servers are required to support password authentication, despite the security weaknesses created by passwords.
- Third-party applications gain overly broad access to the resource-owner's protected resources, leaving resource owners without any ability to restrict duration or access to a limited subset of resources.
- Resource owners cannot revoke access to an individual third-party without revoking access to all third-parties, and must do so by changing their password.

OAuth addresses these issues by introducing an authorization layer and separating the role of the client from that of the resource owner. In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner.

Instead of using the resource owner's credentials to access protected resources, the client obtains an access token - a string denoting a specific scope, duration, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

For example, a web user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo sharing service (resource server), without sharing her username and password with the printing service. Instead, she authenticates directly with a server trusted by the photo sharing service (authorization server) which issues the printing service delegation-specific credentials (access token).

1.1. Roles

OAuth includes four roles working together to grant and provide access to protected resources - access restricted resources which

require authentication to access:

resource owner
: An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

resource server
: The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

client
: An application making protected resource requests on behalf of the resource owner and with its authorization.

authorization server
: The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization.

The interaction between the authorization server and resource server is beyond the scope of this specification. The authorization server may be the same server as the resource server or a separate entity. A single authorization server may issue access tokens accepted by multiple resource servers.

When interacting with the authorization server, the client identifies itself using a set of client credentials which include a client identifier and other authentication attributes. The means through which the client obtains its credentials are beyond the scope of this specification, but typically involve registration with the authorization server.

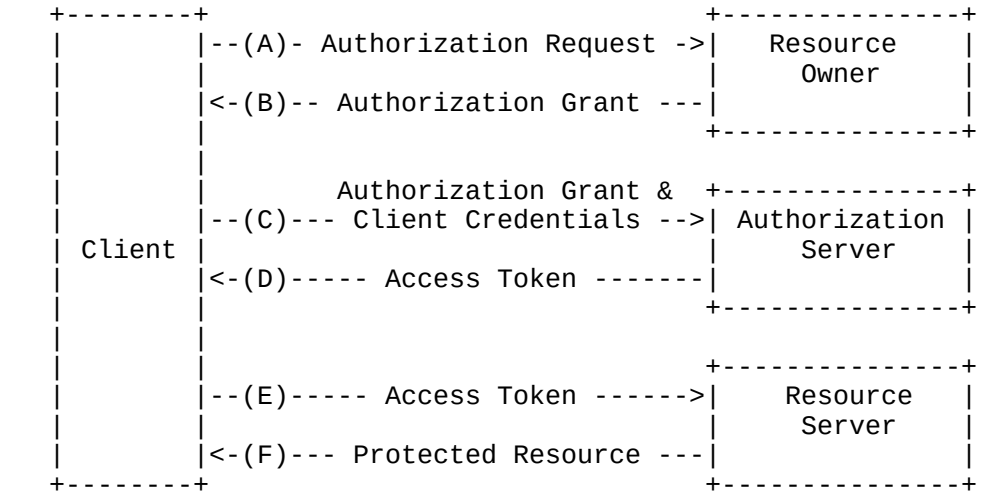


Figure 1: Abstract Protocol Flow

The abstract flow illustrated in Figure 1 describes the interaction between the four roles and includes the following steps:

(A) The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or preferably indirectly via an intermediary such as an authorization server.
(B) The client receives an authorization grant which represents the authorization provided by the resource owner. The authorization grant type depends on the method used by the client and supported by the authorization server to obtain it.
(C) The client requests an access token by authenticating with the authorization server using its client credentials (prearranged between the client and authorization server) and presenting the authorization grant.
(D) The authorization server validates the client credentials and the authorization grant, and if valid issues an access token.
(E) The client requests the protected resource from the resource server and authenticates by presenting the access token.
(F) The resource server validates the access token, and if valid, serves the request.

1.2. Access Token

An access token is a string representing an authorization issued to the client. The string is usually opaque to the client. Tokens represent specific scopes and durations of access, granted by the resource owner, and enforced by the resource server and authorization server.

The token may denote an identifier used to retrieve the authorization information, or self-contain the authorization information in a verifiable manner (i.e. a token string consisting of some data and a signature). Tokens may be pure capabilities. Additional authentication credentials may be required in order for the client to use a token.

The access token provides an abstraction layer, replacing different authorization constructs (e.g. username and password) with a single token understood by the resource server. This abstraction enables issuing access tokens more restrictive than the authorization grant used to obtain them, as well as removing the resource server’s need to understand a wide range of authentication methods.

Access tokens can have different formats, structures, and methods of utilization (e.g. cryptographic properties) based on the resource

server security requirements. Access token attributes and the methods used to access protected resources are beyond the scope of this specification and are defined by companion specifications.

1.3. Authorization Grant

An authorization grant is a general term used to describe the intermediate credentials representing the resource owner authorization, and serves as an abstraction layer. An authorization grant is used by the client to obtain an access token.

1.3.1. Authorization Code

The authorization code is obtained by using an authorization server as an intermediary between the client and resource owner. Instead of requesting authorization directly from the resource owner, the client directs the resource owner to an authorization server (via its user-agent), which in turns directs the resource owner back to the client with the authorization code.

Before directing the resource owner back to the client with the authorization code, the authorization server authenticates the resource owner and obtains authorization. Because the resource owner only authenticates with the authorization server, the resource owner’s credentials are never shared with the client.

The authorization code provides a few important security benefits such as the ability to authenticate the client and issuing the access token directly to the client without potentially exposing it to others, including the resource owner.

1.3.2. Implicit

An implicit grant is issued when the resource owner’s authorization is expressed directly as an access token, without using an intermediate credential. The implicit grant is issued in a similar manner as an authorization code, but instead of the resource owner being redirected back to the client with the authorization code, it is redirected back with an access token and its related attributes.

When issuing an implicit grant, the authorization server cannot verify the identity of the client, and the access token may be exposed to the resource owner or other applications with access to the resource owner’s user-agent.

Implicit grants improve the responsiveness and efficiency of some clients (such as a client implemented as an in-browser application) since it reduces the number of round trip required to obtain an

access token.

1.3.3. Resource Owner Password Credentials

The resource owner password credentials (e.g. a username and password) can be used directly as an authorization grant to obtain an access token. The credentials should only be used when there is a high degree of trust between the resource owner and the client (e.g. its computer operating system or a highly privileged application), and when other authorization grant types are not available (such as an authorization code).

Even though this grant type requires direct client access to the resource owner credentials, the resource owner credentials are used for a single request and are exchanged for an access token. Unlike the HTTP Basic authentication scheme defined in [RFC2617], this grant type eliminates the need for the client to store the resource-owner credentials for future use.

1.3.4. Client Credentials

The client credentials can be used as an authorization grant when the authorization scope is limited to the protected resources under the control of the client, or to protected resources previously arranged with the authorization server. Client credentials are used as an authorization grant typically when the client is acting on its own behalf (the client is also the resource owner).

1.3.5. Extensions

Additional grant types may be defined to provide a bridge between OAuth and other trust frameworks. For example, [I-D.ietf-oauth-saml2-bearer] defines a SAML 2.0 [OASIS.saml-core-2.0-os] bearer assertion grant type, which can be used to obtain an access token.

1.4. Refresh Token

A refresh token is optionally issued by the authorization server to the client together with an access token. The client can use the refresh token to request another access token based on the same authorization, without having to involve the resource owner again, or having to retain the original authorization grant used to obtain the initial access token.

A refresh token is a string representing the authorization granted to the client by the resource owner. The string is usually opaque to the client. The token may denote an identifier used to retrieve the

authorization information, or self-contain the authorization information in a verifiable manner.

The refresh token can be used to obtain a new access token when the current access token expires (access tokens may have a shorter lifetime than authorized by the resource owner), or to obtain additional access tokens with identical or narrower scope.

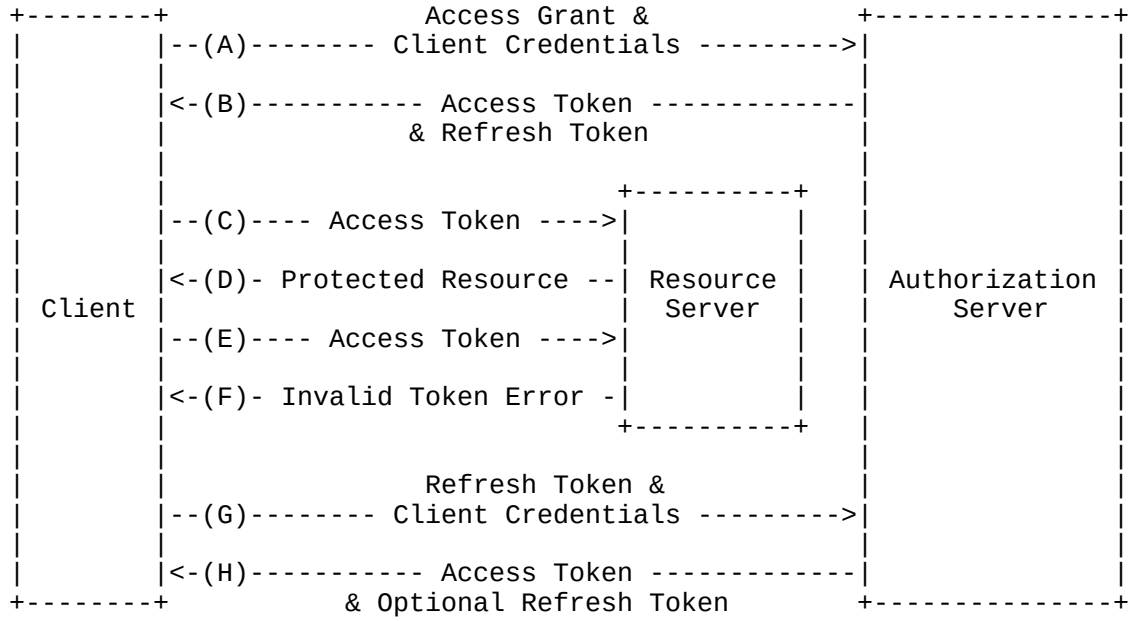


Figure 2: Refreshing an Expired Access Token

The flow illustrated in Figure 2 includes the following steps:

(A) The client requests an access token by authenticating with the authorization server using its client credentials, and presenting an authorization grant.

(B) The authorization server validates the client credentials and the authorization grant, and if valid issues an access token and a refresh token.

(C) The client makes a protected resource requests to the resource server by presenting the access token.

(D) The resource server validates the access token, and if valid, serves the request.
(E) Steps (C) and (D) repeat until the access token expires. If the client knows the access token expired, it skips to step (G), otherwise it makes another protected resource request.
(F) Since the access token is invalid (expired), the resource server returns an invalid token error.
(G) The client requests a new access token by authenticating with the authorization server using its client credentials, and presenting the refresh token.
(H) The authorization server validates the client credentials and the refresh token, and if valid issues a new access token (and optionally, a new refresh token).

1.5. Notational Conventions

The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'MAY', and 'OPTIONAL' in this specification are to be interpreted as described in [RFC2119].

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [I-D.ietf-httpbis-p1-messaging]. Additionally, the following rules are included from [RFC3986]: URI-reference; and from [I-D.ietf-httpbis-p1-messaging]: OWS, RWS, and quoted-string.

Unless otherwise noted, all the protocol parameter names and values are case sensitive.

2. Client Authentication

Client credentials are used to identify and authenticate the client. The client credentials include a client identified - a unique string issued to the client to identify itself to the authorization server. The methods through which the client obtains its client credentials are beyond the scope of this specification.

Due to the nature of some clients, the authorization server SHOULD NOT make assumptions about the confidentiality of client credentials without establishing trust with the client. The authorization server SHOULD NOT issue client credentials to clients incapable of keeping their secrets confidential.

2.1. Client Password Authentication

The client password authentication uses a shared symmetric secret to authenticate the client. The client identifier and password are included in the request using the following parameters:

client_id
: REQUIRED. The client identifier.

client_secret
: REQUIRED. The client password.

For example (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&client_id=s6BhdRkqt3&
client_secret=gX1fBat3bV&code=i1WsRn1uB1&
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

2.2. Other Client Authentication Methods

In cases where client password authentication is not suitable or sufficient, the authorization server MAY support other existing HTTP authentication schemes or define new methods. In addition, the authorization server MAY allow unauthenticated access token requests when the client identity does not matter (e.g. anonymous client) or when the client identity is established via other means.

For example, the authorization server MAY support using the HTTP Basic authentication scheme as defined in [RFC2617] to include the client identifier as the username and client password as the password (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&code=i1WsRn1uB1&
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

When using a method other than client password authentication to exchange an authorization code grant type, the authorization server MUST define a method for mapping the client credentials to the client identifier used to obtain the authorization code.

3. Protocol Endpoints

The authorization process utilizes two endpoints:

- Authorization endpoint - used to obtain authorization from the resource owner via user-agent redirection.
- Token endpoint - used to exchange an authorization grant for an access token, typically with client authentication.

Not every authorization grant flow utilizes both endpoints. Extension grant types MAY define additional endpoints as needed.

3.1. Authorization Endpoint

The authorization endpoint is used to interact with the resource owner and obtain authorization which is expressed explicitly as an authorization code (exchanged for an access token), or implicitly by direct issuance of an access token.

The authorization server MUST first verify the identity of the resource owner. The way in which the authorization server authenticates the resource owner (e.g. username and password login, session cookies) is beyond the scope of this specification.

The location of the authorization endpoint can be found in the service documentation. The endpoint URI MAY include a query component as defined by [RFC3986] section 3, which MUST be retained when adding additional query parameters.

Requests to the authorization endpoint result in user authentication and the transmission of sensitive information. If the response includes an access token, the authorization server MUST require TLS 1.2 as defined in [RFC5246] and MAY support additional transport-layer mechanisms meeting its security requirements. If the response does not include an access token, the authorization server SHOULD require TLS 1.2 and any additional transport-layer mechanism meeting its security requirements.

The authorization server MUST support the use of the HTTP "GET" method for the authorization endpoint, and MAY support the use of the "POST" method as well.

Parameters sent without a value MUST be treated as if they were omitted from the request. The authorization server SHOULD ignore unrecognized request parameters.

3.1.1. Redirection URI

The client directs the resource owner's user-agent to the authorization endpoint and includes a redirection URI to which the authorization server will redirect the user-agent back once authorization has been obtained (or denied). The client MAY omit the redirection URI if one has been established between the client and authorization server via other means, such as during the client registration process.

The redirection URI MUST be an absolute URI and MAY include a query component, which MUST be retained by the authorization server when adding additional query parameters.

The authorization server SHOULD require the client to pre-register their redirection URI or at least certain components such as the scheme, host, port and path. If a redirection URI was registered, the authorization server MUST compare any redirection URI received at the authorization endpoint with the registered URI.

The authorization server SHOULD NOT redirect the user-agent to unregistered or untrusted URIs to prevent the endpoint from being used as an open redirector. If no valid redirection URI is available, the authorization server SHOULD inform the resource owner directly of the error.

3.2. Token Endpoint

The token endpoint is used by the client to obtain an access token by authenticating with the authorization server and presenting its authorization grant. The token endpoint is used with every authorization grant except for the implicit grant type (since an access token is issued directly).

The location of the token endpoint can be found in the service documentation. The endpoint URI MAY include a query component, which MUST be retained when adding additional query parameters.

Since requests to the token endpoint result in the transmission of clear-text credentials (in the HTTP request and response), the authorization server MUST require the use of a transport-layer security mechanism when sending requests to the token endpoints. The authorization server MUST support TLS 1.2 as defined in [RFC5246], and MAY support additional transport-layer mechanisms meeting its security requirements.

The token endpoint requires client authentication as described in Section 2 . The authorization server MAY accept any form of client

authentication meeting its security requirements. The client MUST NOT use more than one authentication method in each request.

The client MUST use the HTTP "POST" method when making access token requests.

Parameters sent without a value MUST be treated as if they were omitted from the request. The authorization server SHOULD ignore unrecognized request parameters.

4. Requesting an Access Token

The client obtains an access token by requesting authorization from the resource owner. The authorization is expressed in the form of an authorization grant which the client exchanges for an access token. OAuth defines four grant types: authorization code, implicit, resource owner password credentials, and client credentials, as well as an extension mechanism for defining additional grant types.

4.1. Authorization Code

The authorization code flow is suitable for clients capable of maintaining their client credentials confidential (for authenticating with the authorization server) such as a client implemented on a secure server. As a redirection-based profile, the client must be capable of interacting with the resource owner’s user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

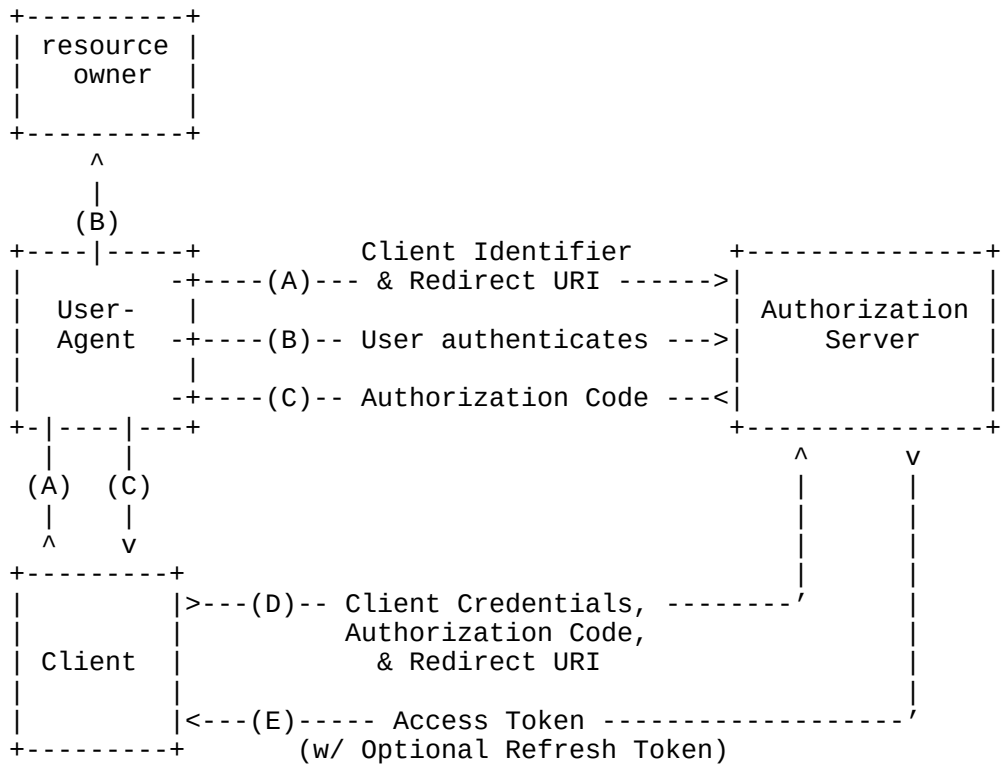


Figure 3: Authorization Code Flow

The flow illustrated in Figure 3 includes the following steps:

(A) The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

(C) Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes an authorization code.

(D) The client requests an access token from the authorization server's token endpoint by authenticating using its client credentials, and includes the authorization code received in the previous step.
(E) The authorization server validates the client credentials and the authorization code and if valid, responds back with an access token.

4.1.1. Authorization Request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format as defined by [W3C.REC-html401-19991224]:

response_type
: REQUIRED. Value MUST be set to "code".

client_id
: REQUIRED. The client identifier as described in Section 2.

redirect_uri
: REQUIRED, unless a redirection URI has been established between the client and authorization server via other means. Described in Section 3.1.1.

scope
: OPTIONAL. The scope of the access request expressed as a list of space-delimited strings. The value is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope.

state
: OPTIONAL. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client.

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using transport-layer security (line breaks are for display purposes only):

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&
    redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

The authorization server validates the request to ensure all required parameters are present and valid. If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

4.1.2. Authorization Response

If the resource owner grants the access request, the authorization server issues an authorization code and delivers it to the client by adding the following parameters to the query component of the redirection URI using the "application/x-www-form-urlencoded" format:

code
: REQUIRED. The authorization code generated by the authorization server. The authorization code SHOULD expire shortly after it is issued to minimize the risk of leaks. The client MUST NOT reuse the authorization code. If an authorization code is used more than once, the authorization server MAY revoke all tokens previously issued based on that authorization code. The authorization code is bound to the client identifier and redirection URI.

state
: REQUIRED if the "state" parameter was present in the client authorization request. Set to the exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?code=i1WsRn1uB1
```

The client SHOULD ignore unrecognized response parameters. The authorization code string size is left undefined by this specification. The clients should avoid making assumptions about code value sizes. The authorization server should document the size of any value is issues.

4.1.2.1. Error Response

If the request fails due to a missing, invalid, or mismatching redirection URI, the authorization server SHOULD inform the resource owner of the error, and MUST NOT redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the query component of the redirection URI using the "application/x-www-form-urlencoded" format:

error
: REQUIRED. A single error code from the following:

  invalid_request
  : The request is missing a required parameter, includes an unsupported parameter or parameter value, or is otherwise malformed.

  invalid_client
  : The client identifier provided is invalid.

  unauthorized_client
  : The client is not authorized to request an authorization code using this method.

  access_denied
  : The resource owner or authorization server denied the request.

  unsupported_response_type
  : The authorization server does not support obtaining an authorization code using this method.

  invalid_scope
  : The requested scope is invalid, unknown, or malformed.

error_description
: OPTIONAL. A human-readable text providing additional information, used to assist in the understanding and resolution of the error occurred.

error_uri
: OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the resource owner with additional information about the error.

state
: REQUIRED if the "state" parameter was present in the client authorization request. Set to the exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb?error=access_denied
```

4.1.3. Access Token Request

The client makes a request to the token endpoint by adding the following parameter using the "application/x-www-form-urlencoded" format in the HTTP request entity-body:

grant_type
: REQUIRED. Value MUST be set to "authorization_code".

code
: REQUIRED. The authorization code received from the authorization server.

redirect_uri
: REQUIRED. The redirection URI used in the initial request.

The client includes its authentication credentials as described in Section 2

For example, the client makes the following HTTP request by including its client credentials via the "client_id" and "client_secret" parameters, and using transport-layer security (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code&client_id=s6BhdRkqt3&
client_secret=gX1fBat3bV&code=i1WsRn1uB1&
redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb
```

The authorization server MUST:

- Validate the client credentials and ensure they match the authorization code.
- Verify that the authorization code and redirection URI are valid and match its stored association.

If the request is valid and authorized, the authorization server

issues an access token and optional refresh token, and responds as described in Section 5.

4.2. Implicit Grant

The implicit grant flow is suitable for clients incapable of maintaining their client credentials confidential (for authenticating with the authorization server) such as client applications residing in a user-agent, typically implemented in a browser using a scripting language such as JavaScript, or native applications. These clients cannot keep client secrets confidential and the authentication of the client is based on the user-agent's same-origin policy.

As a redirection-based profile, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

Unlike the authorization code flow in which the client makes separate requests for authorization and access token, the client receives the access token as the result of the authorization request.

The implicit grant flow does not utilize the client credentials since the client is unable to maintain their confidentiality (the client resides on the resource owner's computer or device which makes the client credentials accessible and exploitable). Because the access token is encoded into the redirection URI, it may be exposed to the resource owner and other applications residing on its computer or device.

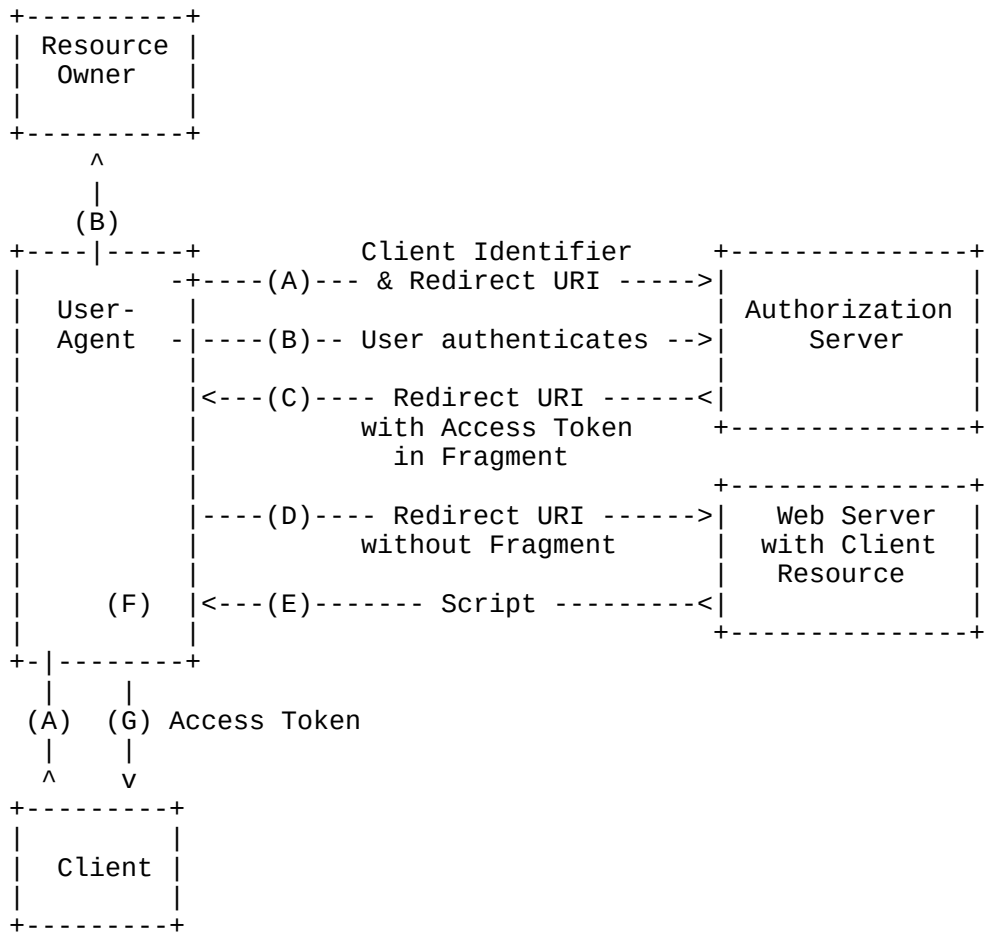


Figure 4: Implicit Grant Flow

The flow illustrated in Figure 4 includes the following steps:

(A) The client initiates the flow by directing the resource owner's user-agent to the authorization endpoint. The client includes its client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the resource owner (via the user-agent) and establishes whether the resource owner grants or denies the client's access request.

(C) Assuming the resource owner grants access, the authorization server redirects the user-agent back to the client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment.
(D) The user-agent follows the redirection instructions by making a request to the web server (does not include the fragment). The user-agent retains the fragment information locally.
(E) The web server returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI including the fragment retained by the user-agent, and extracting the access token (and other parameters) contained in the fragment.
(F) The user-agent executes the script provided by the web server locally, which extracts the access token and passes it to the client.

4.2.1. Authorization Request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

response_type
: REQUIRED. Value MUST be set to "token".

client_id
: REQUIRED. The client identifier as described in Section 2. Due to lack of client authentication, the client identifier alone MUST NOT be relied upon for client identification.

redirect_uri
: REQUIRED, unless a redirection URI has been established between the client and authorization server via other means. Described in Section 3.1.1.

scope
: OPTIONAL. The scope of the access request expressed as a list of space-delimited strings. The value is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope.

state
: OPTIONAL. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client.

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using transport-layer security (line breaks are for display purposes only):

```
GET /authorize?response_type=token&client_id=s6BhdRkqt3&
    redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

The authorization server validates the request to ensure all required parameters are present and valid. If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

4.2.2. Access Token Response

If the resource owner grants the access request, the authorization server issues an access token and delivers it to the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format:

access_token
: REQUIRED. The access token issued by the authorization server.

token_type
: REQUIRED. The type of the token issued as described in Section 7.1. Value is case insensitive.

expires_in
: OPTIONAL. The duration in seconds of the access token lifetime. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated.

scope
: OPTIONAL. The scope of the access request expressed as a list of space-delimited strings. The value is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope. The authorization server SHOULD include the parameter if the requested scope is different from the one requested by the client.

state
: REQUIRED if the "state" parameter was present in the client authorization request. Set to the exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response (URI line breaks are for display purposes only):

```
HTTP/1.1 302 Found
Location: http://example.com/rd#access_token=FJQbwq9&
          token_type=example&expires_in=3600
```

The client SHOULD ignore unrecognized response parameters. The access token string size is left undefined by this specification. The client should avoid making assumptions about value sizes. The authorization server should document the size of any value it issues.

4.2.2.1. Error Response

If the request fails due to a missing, invalid, or mismatching redirection URI, the authorization server SHOULD inform the resource owner of the error, and MUST NOT redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format:

error
: REQUIRED. A single error code from the following:

  invalid_request
  : The request is missing a required parameter, includes an unsupported parameter or parameter value, or is otherwise malformed.

  invalid_client
  : The client identifier provided is invalid.

  unauthorized_client
  : The client is not authorized to request an access token using this method.

access_denied
: The resource owner or authorization server denied the request.

unsupported_response_type
: The authorization server does not support obtaining an access token using this method.

invalid_scope
: The requested scope is invalid, unknown, or malformed.

error_description
: OPTIONAL. A human-readable text providing additional information, used to assist in the understanding and resolution of the error occurred.

error_uri
: OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the resource owner with additional information about the error.

state
: REQUIRED if the "state" parameter was present in the client authorization request. Set to the exact value received from the client.

For example, the authorization server redirects the user-agent by sending the following HTTP response:

```
HTTP/1.1 302 Found
Location: https://client.example.com/cb#error=access_denied
```

4.3. Resource Owner Password Credentials

The resource owner password credentials flow is suitable in cases where the resource owner has a trust relationship with the client, such as its computer operating system or a highly privileged application. The authorization server should take special care when enabling the flow, and only when other flows are not viable.

The flow is suitable for clients capable of obtaining the resource owner credentials (username and password, typically using an interactive form). It is also used to migrate existing clients using direct authentication schemes such as HTTP Basic or Digest authentication to OAuth by converting the stored credentials with an access token.

The method through which the client obtains the resource owner credentials is beyond the scope of this specification. The client MUST discard the credentials once an access token has been obtained.

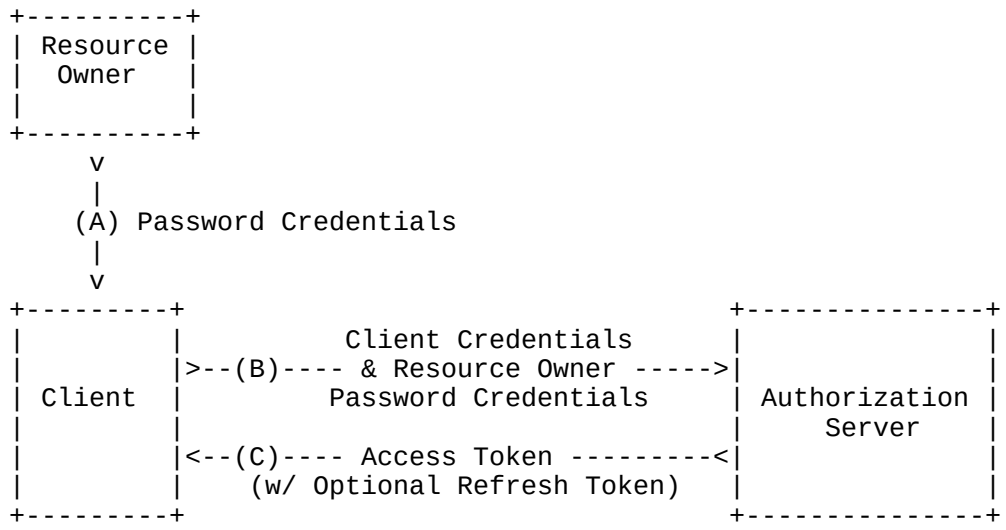


Figure 5: Resource Owner Password Credentials Flow

The flow illustrated in Figure 5 includes the following steps:

(A) The resource owner provides the client with its username and password.
(B) The client requests an access token from the authorization server's token endpoint by authenticating using its client credentials, and includes the credentials received from the resource owner.
(C) The authorization server validates the resource owner credentials and the client credentials and issues an access token.

4.3.1. Access Token Request

The client makes a request to the token endpoint by adding the following parameter using the "application/x-www-form-urlencoded" format in the HTTP request entity-body:

grant_type
: REQUIRED. Value MUST be set to "password".

username
: REQUIRED. The resource owner username.

password
: REQUIRED. The resource owner password.

scope
: OPTIONAL. The scope of the access request expressed as a list of space-delimited strings. The value is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope.

The client includes its authentication credentials as described in Section 2

[[add internationalization consideration for username and password]]

For example, the client makes the following HTTP request by including its client credentials via the "client_id" and "client_secret" parameters, and using transport-layer security (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=password&client_id=s6BhdRkqt3&
client_secret=47HDu8s&username=johndoe&password=A3ddj3w
```

The authorization server MUST:

- Validate the client credentials.
- Validate the resource owner password credentials.

If the request is valid and authorized, the authorization server issues an access token and optional refresh token, and responds as described in Section 5.

4.4. Client Credentials

The client can request an access token using only its client credentials when the client is requesting access to the protected resources under its control, or those of another resource owner which has been previously arranged with the authorization server (the method of which is beyond the scope of this specification).

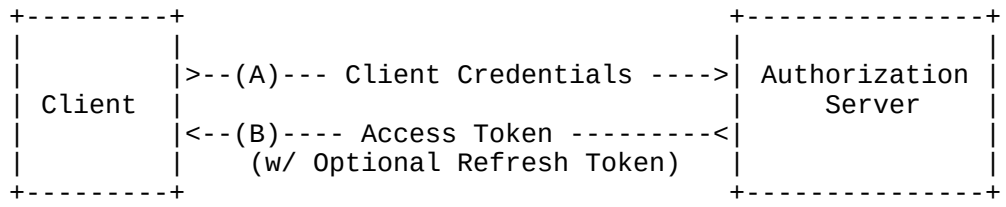


Figure 6: Client Credentials Flow

The flow illustrated in Figure 6 includes the following steps:

(A) The client requests an access token from the token endpoint by authenticating using its client credentials.
(B) The authorization server validates the client credentials and issues an access token.

4.4.1. Access Token Request

The client makes a request to the token endpoint by adding the following parameter using the "application/x-www-form-urlencoded" format in the HTTP request entity-body:

grant_type
: REQUIRED. Value MUST be set to "client_credentials".

scope
: OPTIONAL. The scope of the access request expressed as a list of space-delimited strings. The value is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope.

The client includes its authentication credentials as described in Section 2

For example, the client makes the following HTTP request by including its client credentials via the "client_id" and "client_secret" parameters, and using transport-layer security (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=s6BhdRkqt3&
client_secret=47HDu8s
```

The authorization server MUST validate the client credentials.

If the request is valid and authorized, the authorization server issues an access token and optional refresh token, and responds as described in Section 5.

4.5. Extensions

The client uses an extension grant type by specifying the grant type using an absolute URI (defined by the authorization server) as the value of the "grant_type" parameter of the token endpoint, and by adding any additional parameters necessary.

For example, to request an access token using a SAML 2.0 assertion grant type, the client makes the following HTTP request using transport-layer security (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=http%3A%2F%2Foauth.net%2Fgrant_type%2Fassertion%2F
saml%2F2.0%2Fbearer&assertion=PEFzc2VydGlvbiBJc3N1ZUluc3RhbnQ
[...omitted for brevity...]V0aG5TdGF0ZW1lbnQ-PC9Bc3NlcnRpb24-
```

Client authentication and the scope of the grant are obtained via the assertion as defined by [I-D.ietf-oauth-saml2-bearer].

5. Issuing an Access Token

If the access token request is valid and authorized, the authorization server issues an access token and optional refresh token as described in Section 5.1. If the request failed client authentication or is invalid, the authorization server return an error response as described in Section 5.2.

5.1. Successful Response

The authorization server issues an access token and optional refresh token, and constructs the response by adding the following parameters to the entity body of the HTTP response with a 200 (OK) status code:

access_token
: REQUIRED. The access token issued by the authorization server.

token_type
: REQUIRED. The type of the token issued as described in Section 7.1. Value is case insensitive.

expires_in
: OPTIONAL. The duration in seconds of the access token lifetime. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated.

refresh_token
: OPTIONAL. The refresh token which can be used to obtain new access tokens using the same authorization grant as described in Section 6.

scope
: OPTIONAL. The scope of the access request expressed as a list of space-delimited strings. The value is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope. The authorization server SHOULD include the parameter if the requested scope is different from the one requested by the client.

The parameters are including in the entity body of the HTTP response using the "application/json" media type as defined by [RFC4627]. The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers.

The authorization server MUST include the HTTP "Cache-Control" response header field with a value of "no-store" in any response containing tokens, secrets, or other sensitive information.

For example:

```
  HTTP/1.1 200 OK
  Content-Type: application/json
  Cache-Control: no-store

  {
    "access_token":"SlAV32hkKG",
    "token_type":"example",
    "expires_in":3600,
    "refresh_token":"8xLOxBtZp8",
    "example_parameter":"example-value"
```

```
}
```

The client SHOULD ignore unrecognized response parameters. The sizes of tokens and other values received from the authorization server are left undefined. The client should avoid making assumptions about value sizes. The authorization server should document the size of any value it issues.

5.2. Error Response

If the client provided invalid credentials using an HTTP authentication scheme via the "Authorization" request header field, the authorization server MUST respond with a HTTP 401 (Unauthorized) status code, and include the "WWW-Authenticate" response header field matching the authentication scheme used by the client. Otherwise, the authorization server MUST respond with the HTTP 400 (Bad Request) status code.

The authorization server constructs the response by adding the following parameter to the response:

error
: REQUIRED. A single error code from the following:

  invalid_request
  : The request is missing a required parameter, includes an unsupported parameter or parameter value, repeats a parameter, includes multiple credentials, utilizes more than one mechanism for authenticating the client, or is otherwise malformed.

  invalid_client
  : Client authentication failed (e.g. unknown client, no client credentials included, multiple client credentials included, or unsupported credentials type).

  invalid_grant
  : The provided authorization grant is invalid, expired, revoked, or does not match the redirection URI used in the authorization request.

  unauthorized_client
  : The authenticated client is not authorized to use this authorization grant type.

  unsupported_grant_type
  : The authorization grant type is not supported by the authorization server.

invalid_scope
The requested scope is invalid, unknown, malformed, or exceeds the previously granted scope.

error_description
OPTIONAL. A human-readable text providing additional information, used to assist in the understanding and resolution of the error occurred.

error_uri
OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the resource owner with additional information about the error.

The parameters are including in the entity body of the HTTP response using the "application/json" media type as defined by [RFC4627]. The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers.

For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store

{
  "error":"invalid_request"
}
```

6. Refreshing an Access Token

The client makes a request to the token endpoint by adding the following parameter using the "application/x-www-form-urlencoded" format in the HTTP request entity-body:

grant_type
REQUIRED. Value MUST be set to "refresh_token".

refresh_token
REQUIRED. The refresh token issued along the access token being refreshed.

scope
OPTIONAL. The scope of the access request expressed as a list of space-delimited strings. The value is defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string

adds an additional access range to the requested scope. The requested scope MUST be equal or lesser than the scope originally granted by the resource owner, and if omitted is treated as equal to the previously approved scope.

The client includes its authentication credentials as described in Section 2

For example, the client makes the following HTTP request by including its client credentials via the "client_id" and "client_secret" parameters, and using transport-layer security (line breaks are for display purposes only):

```
POST /token HTTP/1.1
Host: server.example.com
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token&client_id=s6BhdRkqt3&
client_secret=8eSEIpnqmM&refresh_token=n4E9O119d
```

The authorization server MUST validate the client credentials, the refresh token, and verify that the resource owner's authorization is still valid. If valid, the authorization server issues an access token response as described in Section 5.

The authorization server MAY issue a new refresh token, in which case, the client MUST discard the old refresh token and replace it with the new refresh token.

7. Accessing Protected Resources

The client accesses protected resources by presenting the access token to the resource server. The resource server MUST validate the access token and ensure it has not expired and that its scope covers the requested resource. The methods used by the resource server to validate the access token are beyond the scope of this specification, but generally involve an interaction or coordination between the resource server and the authorization server.

The method in which the client utilized the access token to authenticate with the resource server depends on the type of access token issued by the authorization server. Typically, it involves using the HTTP "Authorization" request header field with an authentication scheme defined by the access token type specification.

7.1. Access Token Types

The access token type provides the client with the information required to successfully utilize the access token to make a protected resource request (along with type-specific attributes).

For example, the "bearer" token type defined in [I-D.ietf-oauth-v2-bearer] is utilized by simply including the access token string in the request:

```
GET /resource/1 HTTP/1.1
Host: example.com
Authorization: BEARER h480djs93hd8
```

while the "mac" token type defined in [I-D.hammer-oauth-v2-mac-token] is utilized by issuing a token secret together with the access token which is used to sign certain components of the HTTP requests:

```
GET /resource/1 HTTP/1.1
Host: example.com
Authorization: MAC token="h480djs93hd8",
                   timestamp="137131200",
                   nonce="dj83hs9s",
                   signature="kDZvddkndxvhGRXZhvuDjEWhGeE="
```

Each access token type definition specifies the additional attributes (if any) sent to the client together with the "access_token" response parameter. It also defines the HTTP authentication method used to include the access token when making a protected resource request.

8. Extensibility

8.1. Defining Access Token Types

Access token types can be defined in one of two ways: registered in the access token type registry (following the procedures in Section 10.1), or use the "x_" type name prefix.

Types utilizing the "x_" name prefix MUST be limited to vendor-specific implementations that are not commonly applicable, and are specific to the implementation details of the resource server where they are used. If a vendor-specific type requires additional vendor-specific token response parameters, they MUST also use the "x_" name

prefix.

All other types MUST be registered, and MUST NOT use the "x_" type name prefix. Type names MUST conform to the type-name ABNF. If the type definition includes a new HTTP authentication scheme, the type name SHOULD be identical to the authentication scheme name (as defined by [RFC2617]).

```
type-name  = 1*name-char
name-char  = "-" / "." / "_" / DIGIT / ALPHA
```

8.2. Defining New Endpoint Parameters

New request or response parameters for use with the authorization endpoint or the token endpoint can be added in one of two ways: registered in the parameters registry (following the procedures in Section 10.2), or use the "x_" parameter name prefix.

Parameters utilizing the "x_" parameter name prefix MUST be limited to vendor-specific extensions that are not commonly applicable, and are specific to the implementation details of the authorization server where they are used. All other new parameters MUST be registered, and MUST NOT use the "x_" parameter name prefix.

Parameter names MUST conform to the param-name ABNF, and parameter values syntax MUST be well-defined (e.g., using ABNF, or a reference to the syntax of an existing parameter).

```
param-name  = 1*name-char
name-char   = "-" / "." / "_" / DIGIT / ALPHA
```

8.3. Defining New Authorization Grant Types

New authorization grant types can be defined by assigning them a unique URI for use with the "grant_type" parameter. If the extension grant type requires additional token endpoint parameters, they MUST be registered in the OAuth parameters registry as described by Section 10.2.

9. Security Considerations

[[TBD]]

10. IANA Considerations

10.1. The OAuth Access Token Type Registry

This specification establishes the OAuth access token type registry.

Access token types are registered on the advice of one or more Designated Experts (appointed by the IESG or their delegate), with a Specification Required (using terminology from [RFC5226]). However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests should be sent to the [TBD]@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for access toke type: example"). [[Note to RFC-EDITOR: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: oauth-ext-review.]]

Before a period of 14 days has passed, the Designated Expert(s) will either approve or deny the registration request, communicating this decision both to the review list and to IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the iesg@iesg.org mailing list) for resolution.

10.1.1. Registration Template

Type name:
: The name requested (e.g., "example").

Additional Token Endpoint Response Parameters:
: Additional response parameters returned together with the "access_token" parameter. New parameters MUST be separately registered in the OAuth parameters registry as described by Section 10.2.

HTTP Authentication Scheme(s):
: The HTTP authentication scheme name(s), if any, used to authenticate protected resources requests using access token of this type.

Change controller:
: For standards-track RFCs, state "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, e-mail address, home page URI) may also be included.

Specification document(s):
: Reference to document that specifies the parameter, preferably including a URI that can be used to retrieve a copy of the document. An indication of the relevant sections may also be included, but is not required.

10.2. The OAuth Parameters Registry

This specification establishes the OAuth parameters registry.

Additional parameters for inclusion in the authorization endpoint request, the authorization endpoint response, the token endpoint request, or the token endpoint response, are registered on the advice of one or more Designated Experts (appointed by the IESG or their delegate), with a Specification Required (using terminology from [RFC5226]). However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests should be sent to the [TBD]@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for parameter: example"). [[Note to RFC-EDITOR: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: oauth-ext-review.]]

Before a period of 14 days has passed, the Designated Expert(s) will either approve or deny the registration request, communicating this decision both to the review list and to IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the iesg@iesg.org mailing list) for resolution.

10.2.1. Registration Template

Parameter name:
: The name requested (e.g., "example").

Parameter usage location:
: The location(s) where parameter can be used. The possible locations are: authorization request, authorization response, token request, or token response.

Change controller:
: For standards-track RFCs, state "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, e-mail address, home page URI) may also be included.

Specification document(s):
Reference to document that specifies the parameter, preferably including a URI that can be used to retrieve a copy of the document. An indication of the relevant sections may also be included, but is not required.

10.2.2. Initial Registry Contents

The OAuth Parameters Registry's initial contents are:

- Parameter name: client_id
- Parameter usage location: authorization request, token request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: client_secret
- Parameter usage location: token request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: response_type
- Parameter usage location: authorization request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: redirect_uri
- Parameter usage location: authorization request, token request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: scope
- Parameter usage location: authorization request, authorization response, token request, token response
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: state
- Parameter usage location: authorization request, authorization response
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: code
- Parameter usage location: authorization response, token request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: error_description
- Parameter usage location: authorization response, token response
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: error_uri
- Parameter usage location: authorization response, token response
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: grant_type
- Parameter usage location: token request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: access_token
- Parameter usage location: authorization response, token response
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: token_type
- Parameter usage location: authorization response, token response
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: expires_in
- Parameter usage location: authorization response, token response
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: username
- Parameter usage location: token request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: password
- Parameter usage location: token request
- Change controller: IETF
- Specification document(s): [[this document]]

- Parameter name: refresh_token
- Parameter usage location: token request, token response
- Change controller: IETF
- Specification document(s): [[this document]]

Appendix A. Examples

[[TBD]]

Appendix B. Contributors

The following people contributed to preliminary versions of this document: Blaine Cook (BT), Brian Eaton (Google), Yaron Goland (Microsoft), Brent Goldman (Facebook), Raffi Krikorian (Twitter), Luke Shepard (Facebook), and Allen Tom (Yahoo!). The content and concepts within are a product of the OAuth community, WRAP community, and the OAuth Working Group.

The OAuth Working Group has dozens of very active contributors who proposed ideas and wording for this document, including:

Michael Adams, Andrew Arnott, Dirk Balfanz, Brian Campbell, Leah Culver, Bill de hOra, Brian Ellin, Igor Faynberg, George Fletcher, Tim Freeman, Evan Gilbert, Kristoffer Gronowski, Justin Hart, Phil Hunt, Mike Jones, John Kemp, Chasen Le Hara, Torsten Lodderstedt, Alastair Mair, Eve Maler, James Manger, Laurence Miao, Chuck Mortimore, Justin Richer, Peter Saint-Andre, Nat Sakimura, Rob Sayre, Marius Scurtescu, Naitik Shah, Justin Smith, Jeremy Suriel, Christian Stuebner, Paul Tarjan, Franklin Tse, and Nick Walker.

Appendix C. Acknowledgements

[[Add OAuth 1.0a authors + WG contributors]]

Appendix D. Document History

[[to be removed by RFC editor before publication as an RFC]]

-12

- Complete restructure with lots of new prose.
- Removed terminology and expanded terms in overview.
- Changed assertions to extensions and added informative reference to the SAML 2.0 extension.
- Renamed access grant to authorization grant.
- Clarified 'token_type' as case insensitive.
- Authorization endpoint requires TLS when an access token is issued.

- Removed client assertion credentials, mandatory HTTP Basic authentication support for client credentials, WWW-Authenticate header, and the OAuth2 authentication scheme.
- Changed implicit grant (aka user-agent flow) error response from query to fragment.
- Removed the 'redirect_uri_mismatch' error code since in such a case, the authorization server must not send the error back to the client.
- Added parameter registration for all parameters in this specification.
- Defined access token type registry.

-11

- Many editorial changes. Fixed user authorization section structure. Removed unused normative references. Adjusted language regarding single use of authorization codes.
- Fixed header ABNF.
- Change access token description from shared symmetric secret to password.
- Moved access grant 'none' to a separate section, renamed to 'client_credentials'.
- Demoted the HTTP status code requirement from MUST to SHOULD in protected resource response error.
- Removed 'expired_token' error code.
- Moved all the 'code_and_token' parameter to the fragment (from code being in the query).
- Removed 'assertion_type' parameter (moved to 'grant_type').
- Added note about redirecting to invalid redirection URIs (open redirectors).
- Removed bearer token section, added new required 'token_type' parameter with extensibility.
- 'error-uri' parameter value changed to absolute URI.
- OAuth 2.0 HTTP authentication scheme name changed to 'OAuth2'.
- Dropped the 'WWW-Authenticate' header field 'realm' parameter.
- Removed definition of access token characters.
- Added instructions for dealing with error and an invalid redirection URI.

-10

- Fixed typos. Many editorial changes. Rewrote introduction. removed terminology grouping.
- Allowed POST for resource owner authorization endpoint.
- Fixed token endpoint to not require client authentication.
- Made URI query and POST body 'oauth_token' parameter optional.

- Moved all parameter names and values to use underscores.
- Changed 'basic_credentials' to 'password', 'invalid_client_credentials' and 'invalid_client_id' to 'invalid_client'.
- Added note that access token requests without an access grant should not include a refresh token.
- Changed scheme name from 'Token' to 'OAuth', simplified request format to simple string for token instead of key=value pair (still supported for extensions).
- Defined permitted access token string characters (suitable for inclusion in an HTTP header).
- Added a note about conflicts with previous versions.
- Moved 'client_id' definition from client authentication to access token endpoint.
- Added definition for 'access grant'.

-09

- Fixed typos, editorial changes.
- Added token expiration example.
- Added scope parameter to resource owner authorization endpoint response.
- Added note about parameters with empty values (same as omitted).
- Changed parameter values to use '-' instead of '_'. Parameter names still use '_'.
- Changed authorization endpoint client type to response type with values: code, token, and both.
- Complete cleanup of error codes. Added support for error description and URI.
- Add initial extensibility support.

-08

- Renamed verification code to authorization code.
- Revised terminology, structured section, added new terms.
- Changed flows to profiles and moved to introduction.
- Added support for access token rescoping.
- Cleaned up client credentials section.
- New introduction overview.
- Added error code for invalid username and password, and renamed error code to be more consistent.
- Added access grant type parameter to token endpoint.

-07

- Major rewrite of entire document structure.

- Removed device profile.
- Added verification code support to user-agent flow.
- Removed multiple formats support, leaving JSON as the only format.
- Changed assertion "assertion_format" parameter to "assertion_type".
- Removed "type" parameter from token endpoint.

-06

- Editorial changes, corrections, clarifications, etc.
- Removed conformance section.
- Moved authors section to contributors appendix.
- Added section on native applications.
- Changed error response to use the requested format. Added support for HTTP "Accept" header.
- Flipped the order of the web server and user-agent flows.
- Renamed assertion flow "format" parameter name to "assertion_format" to resolve conflict.
- Removed the term identifier from token definitions. Added a cryptographic token definition.
- Added figure titles.
- Added server response 401 when client tried to authenticate using multiple credentials.
- Clarified support for TLS alternatives, and added requirement for TLS 1.2 support for token endpoint.
- Removed all signature and cryptography.
- Removed all discovery.
- Updated HTML4 reference.

-05

- Corrected device example.
- Added client credentials parameters to the assertion flow as OPTIONAL.
- Added the ability to send client credentials using an HTTP authentication scheme.
- Initial text for the "WWW-Authenticate" header (also added scope support).
- Change authorization endpoint to resource owner endpoint.
- In the device flow, change the "user_uri" parameter to "verification_uri" to avoid confusion with the resource owner endpoint.
- Add "format" request parameter and support for XML and form-encoded responses.

-04

- Changed all token endpoints to use "POST"
- Clarified the authorization server's ability to issue a new refresh token when refreshing a token.
- Changed the flow categories to clarify the autonomous group.
- Changed client credentials language not to always be server-issued.
- Added a "scope" response parameter.
- Fixed typos.
- Fixed broken document structure.

-03

- Fixed typo in JSON error examples.
- Fixed general typos.
- Moved all flows sections up one level.

-02

- Removed restriction on "redirect_uri" including a query.
- Added "scope" parameter.
- Initial proposal for a JSON-based token response format.

-01

- Editorial changes based on feedback from Brian Eaton, Bill Keenan, and Chuck Mortimore.
- Changed device flow "type" parameter values and switch to use only the token endpoint.

-00

- Initial draft based on a combination of WRAP and OAuth 1.0a.

11. References

11.1. Normative References

[I-D.ietf-httpbis-p1-messaging]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H., Masinter, L., Leach, P., Berners-Lee, T., and J. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", draft-ietf-httpbis-p1-messaging-09 (work in progress), March 2010.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2828] Shirey, R., "Internet Security Glossary", RFC 2828, May 2000.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, January 2001.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.

[W3C.REC-html401-19991224] Hors, A., Raggett, D., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999,

<http://www.w3.org/TR/1999/REC-html401-19991224>.

11.2. Informative References

[I-D.hammer-oauth-v2-mac-token]
Hammer-Lahav, E., "HTTP Authentication: MAC Authentication", draft-hammer-oauth-v2-mac-token-01 (work in progress), January 2011.

[I-D.ietf-oauth-saml2-bearer]
Campbell, B. and C. Mortimore, "SAML 2.0 Bearer Assertion Grant Type Profile for OAuth 2.0", draft-ietf-oauth-saml2-bearer-00 (work in progress), December 2010.

[I-D.ietf-oauth-v2-bearer]
Jones, M., Hardt, D., and D. Recordon, "The OAuth 2.0 Protocol: Bearer Tokens", draft-ietf-oauth-v2-bearer-01 (work in progress), December 2010.

[OASIS.saml-core-2.0-os]
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.

Authors' Addresses

Eran Hammer-Lahav (editor)
Yahoo!

Email: eran@hueniverse.com
URI: http://hueniverse.com

David Recordon
Facebook

Email: dr@fb.com
URI: http://www.davidrecordon.com/

Dick Hardt
Microsoft

Email: dick.hardt@gmail.com
URI: http://dickhardt.org/