

OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 11, 2013

T. Lodderstedt, Ed.
Deutsche Telekom AG
S. Dronia

M. Scurtescu
Google
January 7, 2013

Token Revocation
draft-ietf-oauth-revocation-04

Abstract

This document proposes an additional endpoint for OAuth authorization servers, which allows clients to notify the authorization server that a previously obtained refresh or access token is no longer needed. This allows the authorization server to cleanup security credentials. A revocation request will invalidate the actual token and, if applicable, other tokens based on the same authorization.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119](#) [[RFC2119](#)].

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 11, 2013.

Copyright Notice

Copyright (c) 2013 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
2.	Token Revocation	3
2.1.	Cross-Origin Support	5
3.	Implementation Note	5
4.	Acknowledgements	6
5.	IANA Considerations	6
6.	Security Considerations	6
7.	References	7
7.1.	Normative References	7
7.2.	Informative References	7
	Authors' Addresses	8

1. Introduction

The OAuth 2.0 core specification [[RFC6749](#)] defines several ways for a client to obtain refresh and access tokens. This specification supplements the core specification with a mechanism to revoke both types of tokens. A token is a string representing an authorization issued by the resource owner to the client. A revocation request will invalidate the actual token and, if applicable, other tokens based on the same authorization and the authorization itself.

From an end-user's perspective, OAuth is often used to log into a certain site or app. This revocation mechanism allows a client to invalidate its tokens if the end-user logs out, changes identity, or uninstalls the respective app. Notifying the authorization server that the token is no longer needed allows the authorization server to clean up data associated with that token (e.g. session data) and the underlying authorization. This behavior prevents a situation where there is still a valid authorization for a particular client which the end user is not aware of. This way, token revocation prevents abuse of abandoned tokens and facilitates a better end-user experience since invalidated authorization will no longer turn up in a list of authorizations the authorization server might present to the end-user.

2. Token Revocation

The client requests the revocation of a particular token by making an HTTP POST request to the token revocation endpoint. The location of the token revocation endpoint can be found in the authorization server's documentation. The token endpoint URI MAY include a query component.

Implementations **MUST** support the revocation of refresh tokens and **SHOULD** support the revocation of access tokens (see Implementation Note).

Since requests to the token revocation endpoint result in the transmission of plain text credentials in the HTTP request, the authorization server **MUST** require the use of a transport-layer security mechanism when sending requests to the token revocation endpoints. The authorization server **MUST** support TLS 1.0 ([[RFC2246](#)]), **SHOULD** support TLS 1.2 ([[RFC5246](#)]) and its future replacements, and **MAY** support additional transport-layer mechanisms meeting its security requirements.

The client constructs the request by including the following parameters using the "application/x-www-form-urlencoded" format in

the HTTP request entity-body:

token REQUIRED. The token that the client wants to get revoked.

Note: the authorization server is supposed to detect the token type automatically.

The client also includes its authentication credentials as described in [Section 2.3. of \[RFC6749\]](#).

For example, a client may request the revocation of a refresh token with the following request (line breaks are for display purposes only):

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

```
token=45ghiukldjahdnhdauz&
```

The authorization server first validates the client credentials (in case of a confidential client) and then verifies whether the token was issued to the client making the revocation request. If this validation fails, the request is refused and the client is informed of the error by the authorization server as described below.

In the next step, the authorization server invalidates the token. The client MUST NOT use this token again after revocation.

Depending on the authorization server's revocation policy, the revocation of a particular token may cause the revocation of related tokens and the underlying authorization. If the particular token is a refresh token and the authorization server supports the revocation of access tokens, then the authorization server SHOULD also invalidate all access tokens based on the same authorization (see Implementation Note).

The authorization server indicates a successful processing of the request by a HTTP status code 200. The error presentation conforms to the definition in [section 5.2 of \[RFC6749\]](#).

The following additional error codes are defined for the token revocation endpoint:

unsupported_token_type The authorization server does not support the revocation of the presented token type. I.e. the client tried to revoke an access token on a server not supporting this feature.

`invalid_token` The presented token is invalid.

2.1. Cross-Origin Support

The revocation end-point SHOULD support CORS [[W3C.WD-cors-20120403](#)] if it is aimed at use in combination with user-agent-based applications.

In addition, for interoperability with legacy user-agents, it MAY offer JSONP [[jsonp](#)] by allowing GET requests with an additional parameter:

`callback` OPTIONAL. The qualified name of a JavaScript function.

For example, a client may request the revocation of an access token with the following request (line breaks are for display purposes only):

```
https://example.com/revoke?token=agabcdefdddaafdd&
callback=package.myCallback
```

Successful response:

```
package.myCallback();
```

Error response:

```
package.myCallback({"error":"invalid_token"});
```

Clients should be aware that when relying on JSONP, a malicious revocation end-point may attempt to inject malicious code into the client.

3. Implementation Note

OAuth 2.0 allows deployment flexibility with respect to the style of access tokens. The access tokens may be self-contained so that an resource server needs no further interaction with an authorization server issuing these tokens to perform an authorization decision of the client requesting access to a protected resource. A system design may, however, instead use access tokens that are handles referring to authorization data stored at the authorization server. This consequently requires a resource server to issue a request to the respective authorization server to retrieve the content of the access token every time a client presents an access token.

While these are not the only options they illustrate the implications

for revocation. In the latter case the authorization server is able to revoke an access token previously issued to a client when the resource server relays a received access token. In the former case some (currently non-standardized) backend interaction between the authorization server and the resource server may be used when immediate access token revocation is desired. Another design alternative is to issue short-lived access tokens, which can be refreshed at any time using the corresponding refresh tokens. This allows the authorization server to impose a limit on the time revoked access tokens are in use.

Which approach of token revocation is chosen will depend on the overall system design and on the application service provider's risk analysis. The cost of revocation in terms of required state and communication overhead is ultimately the result of the desired security properties.

4. Acknowledgements

We would like to thank Peter, Mauritius, Amanda Anganes, Mark Wubben, Hannes Tschofenig, Michiel de Jong, Doug Foiles, Paul Madsen, George Fletcher, Sebastian Ebling, Christian Stuebner, Brian Campbell, Igor Faynberg, Lukas Rosenstock, and Justin Richer for their valuable feedback.

5. IANA Considerations

This draft includes no request to IANA.

6. Security Considerations

If the authorization server does not support access token revocation, access tokens will not be immediately invalidated when the corresponding refresh token is revoked. Deployments **MUST** take this into account when conducting their security risk analysis.

Cleaning up tokens using revocation contributes to overall security and privacy since it reduces the likelihood for abuse of abandoned tokens. This specification in general does not intend to provide countermeasures against token theft and abuse. For a discussion of respective threats and countermeasures, consult the security considerations given in [section 10](#) of the OAuth core specification [[RFC6749](#)] and the OAuth threat model document [[I-D.ietf-oauth-v2-threatmodel](#)].

Malicious clients could attempt to use the new endpoint to launch denial of service attacks on the authorization server. Appropriate countermeasures, which should be in place for the token endpoint as well, MUST be applied to the revocation endpoint (see [\[I-D.ietf-oauth-v2-threatmodel\]](#), section 4.4.1.11).

A malicious client may attempt to guess valid tokens on this endpoint by making revocation requests against potential token strings. According to this specification, a client's request must contain a valid `client_id`, in the case of a public client, or valid client credentials, in the case of a confidential client. The token being revoked must also belong to the requesting client. If an attacker is able to successfully guess a public client's `client_id` and one of their tokens, or a private client's credentials and one of their tokens, they could do much worse damage by using the token elsewhere than by revoking it. If they chose to revoke the token, the legitimate client will lose its authorization and will need to prompt the user again. No further damage is done and the guessed token is now worthless.

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", [RFC 2246](#), January 1999.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), August 2008.
- [RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", [RFC 6749](#), October 2012.

7.2. Informative References

- [I-D.ietf-oauth-v2-threatmodel]
Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", [draft-ietf-oauth-v2-threatmodel-08](#) (work in progress), October 2012.
- [W3C.WD-cors-20120403]
Kesteren, A., "Cross-Origin Resource Sharing", World Wide Web Consortium LastCall WD-cors-20120403, April 2012,

<<http://www.w3.org/TR/2012/WD-cors-20120403>>.

[jsonp] Ippolito, B., "Remote JSON - JSONP", December 2005.

Authors' Addresses

Torsten Lodderstedt (editor)
Deutsche Telekom AG

Email: torsten@lodderstedt.net

Stefanie Dronia

Email: sdronia@gmx.de

Marius Scurtescu
Google

Email: mscurtescu@google.com

