OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 28, 2013

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

M. Scurtescu
Google

November 24, 2012

Token Revocation
draft-ietf-oauth-revocation-03

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 access grant.

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 May 28, 2013.

Copyright Notice

Copyright (c) 2012 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 7

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 the external representation of an access grant issued by a resource owner to a particular client. A revocation request will invalidate the actual token and, if applicable, other tokens based on the same access grant and the access grant itself.

From an end-user's perception, 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 access grant. This behavior prevents a situation where there is still a valid access grant 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 access grants will no longer turn up in a list of access grants 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=45ghiukldjahdnhzdauz&
```

The authorization server first validates the client credentials (in case of a confidential client) and verifies whether the client is authorized to revoke the particular token. These checks are used to validate whether the token being presented has been issued to the client presenting it.

In the next step, the authorization server invalidates the token and the respective access grant. 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 access grant (see Implementation Note).

The client MUST NOT use the token again after revocation.

The authorization server indicates a successful processing of the request by a HTTP status code 200. Status code 401 indicates a failed client authentication, whereas a status code 403 is used if the client is not authorized to revoke the particular token. For all other error conditions, a status code 400 is used along with an error response as defined in section 5.2. of [RFC6749]. The following 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 revokation 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 The qualified name of a JavaScript function.

Example request:

```
https://example.com/revoke?token=45ghiukldjahdnhzdauz&
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 revokation end-point may attempt to inject malicious code into the client.

3. Implementation Note

Depending on the authorization server's token design, revocation of access tokens might be a costly process. For example, revocation of self-contained access tokens requires (time-consuming) backend calls between resource and authorization server on every request to the resource server or to push notifications from the authorization server to the affected resource servers. Alternatively, authorization servers may choose to issue short living access tokens, which can be refreshed at any time using the corresponding refresh tokens. In this case, a client would revoke the refresh token and access tokens issued based on this particular refresh token are at most valid until expiration. Whether this is an viable option or whether access token revocation is required should be decided based on the service provider's risk analysis.

4. Acknowledgements

We would like to thank 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 in 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.

A malicious client may attempt to guess valid tokens on this endpoints. As a pre-requisite, the client either requires a valid client_id of a public client or the credentials of a confidential client. An sucessful attempt would result in the revocation of the respective token, thus causing the legitimate client to lose its authorization. The malicious client does not gain further advantages.

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