Network Working Group	M. Jones, Ed.	
Internet-Draft	Microsoft	
Intended status: Standards Track	D. Hardt	
Expires: June 4, 2011	independent	
	D. Recordon	
	Facebook	
	December 1, 2010	

TOC

The OAuth 2.0 Protocol: Bearer Tokens
draft-ietf-oauth-v2-bearer-01

Abstract

This specification describes how to use bearer tokens when accessing OAuth 2.0 protected resources.

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 June 4, 2011.

Table of Contents

1. Introduction

TOC

OAuth enables clients to access protected resources by obtaining an access token, which is defined in [OAuth2] (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, “The OAuth 2.0 Protocol,” 2010.) as "a string representing an access authorization issued to the client", rather than using the resource owner's credentials.

Tokens are issued to 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. This specification describes how to make protected resource requests by treating an OAuth access token as a bearer token.

This specification defines the use of bearer tokens with OAuth over HTTP (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.) [RFC2616] using TLS (Rescorla, E., “HTTP Over TLS,” May 2000.) [RFC2818]. Other specifications may extend it for use with other transport protocols.

TOC

1.1. Notational Conventions

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 [RFC2119] (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.).

This document uses the Augmented Backus-Naur Form (ABNF) notation of [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," March 2010.). Additionally, the following rules are included from [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," June 1999.): auth-param; and from [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," March 2010.): RWS.

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

1.2. Terminology

TOC

All terms are as defined in The OAuth 2.0 Protocol (Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, "The OAuth 2.0 Protocol," 2010.) [OAuth2].

1.3. Overview

TOC

OAuth provides a method for clients to access a protected resource on behalf of a resource owner. Before a client can access a protected resource, it must first obtain authorization (access grant) from the resource owner, then exchange the access grant for an access token (representing the grant's scope, duration, and other attributes). The client accesses the protected resource by presenting the access token to the resource server.

The access token provides an abstraction layer, replacing different authorization constructs (e.g. username and password, assertion) for a single token understood by the resource server. This abstraction enables issuing access tokens valid for a short time period, as well as removing the resource server's need to understand a wide range of authentication schemes.

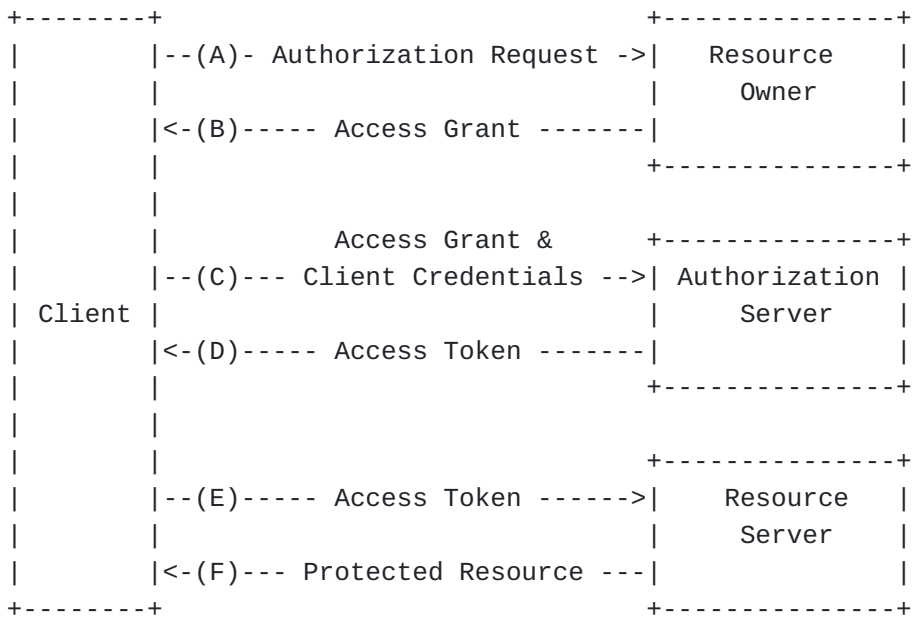

Figure 1: Abstract Protocol Flow

The abstract flow illustrated in Figure 1 (Abstract Protocol Flow) describes the overall OAuth 2.0 protocol architecture. The following steps are specified within this document:

> E) The client makes a protected resource request to the resource server by presenting the access token.
>
> F) The resource server validates the access token, and if valid, serves the request.

2. Authenticated Requests

TOC

Clients make authenticated token requests using the Authorization request header field. Resource servers MUST accept authenticated requests using the OAuth2 HTTP authentication scheme as described in Section 2.1 (The Authorization Request Header Field), and MAY support additional methods.
Alternatively, clients MAY attempt to include the access token in the HTTP body when using the application/x-www-form-urlencoded content type as described in Section 2.2 (Form-Encoded Body Parameter) or using the HTTP request URI in the query component as described in Section 2.3

(URI Query Parameter). Resource servers MAY support these alternative methods.
Clients SHOULD only use the request body or URI when the Authorization request header field is not available, and MUST NOT use more than one method to transport the token in each request. Because of the Security Considerations (Security Considerations) associated with the URI method, it SHOULD only be used if no other method is feasible.

2.1. The Authorization Request Header Field

TOC

The Authorization request header field is used by clients to make authenticated token requests. The client uses the OAuth2 authentication scheme to include the access token in the request.
For example:

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: OAuth2 vF9dft4qmT
```

The Authorization header field uses the framework defined by [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication,” June 1999.) as follows:

```
credentials    = "OAuth2" RWS access-token [ RWS 1#auth-param ]
access-token   = 1*( quoted-char / <"> )

quoted-char    =   "!" / "#" / "$" / "%" / "&" / "'" / "("
                 / ")" / "*" / "+" / "-" / "." / "/" / DIGIT
                 / ":" / "<" / "=" / ">" / "?" / "@" / ALPHA
                 / "[" / "]" / "^" / "_" / "`" / "{" / "|"
                 / "}" / "~" / "\" / "," / ";"
```

2.2. Form-Encoded Body Parameter

TOC

When including the access token in the HTTP request entity-body, the client adds the access token to the request body using the oauth_token parameter. The client can use this method only if the following REQUIRED conditions are met:

*The HTTP request entity-body is single-part.

*The entity-body follows the encoding requirements of the application/x-www-form-urlencoded content-type as defined by [W3C.REC-html401-19991224] (Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification," December 1999.).

*The HTTP request entity-header includes the Content-Type header field set to application/x-www-form-urlencoded.

*The HTTP request method is one for which a body is permitted to be present in the request. In particular, this means that the GET method MAY NOT be used.

The entity-body can include other request-specific parameters, in which case, the oauth_token parameters SHOULD be appended following the request-specific parameters, properly separated by an & character (ASCII code 38).
For example, the client makes the following HTTP request using transport-layer security:

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

oauth_token=vF9dft4qmT
```

The application/x-www-form-urlencoded method should typically only be used in application contexts where participating browsers do not have access to the Authorization request header field.

2.3. URI Query Parameter

TOC

When including the access token in the HTTP request URI, the client adds the access token to the request URI query component as defined by [RFC3986] (Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," January 2005.) using the oauth_token parameter.
For example, the client makes the following HTTP request using transport-layer security:

```
GET /resource?oauth_token=vF9dft4qmT HTTP/1.1
Host: server.example.com
```

The HTTP request URI query can include other request-specific parameters, in which case, the oauth_token parameters SHOULD be appended following the request-specific parameters, properly separated by an & character (ASCII code 38).
For example:

```
http://example.com/resource?x=y&oauth_token=vF9dft4qmT
```

Because of the Security Considerations (Security Considerations) associated with the URI method, it SHOULD only be used if no other method is feasible.

---

3. Security Considerations

TOC

This section describes the relevant security threats regarding token handling when using bearer tokens and describes how to mitigate these threats.

3.1. Security Threats

TOC

The following list presents several common threats against protocols utilizing some form of tokens. This list of threats is based on NIST Special Publication 800-63 [NIST800-63] (Burr, W., Dodson, D., Perlner, R., Polk, T., Gupta, S., and E. Nabbus, "NIST Special Publication 800-63-1, INFORMATION SECURITY," December 2008.). Since this document builds on the OAuth 2.0 specification, we exclude a discussion of threats that are described there or in related documents.

> **Token manufacture/modification:** An attacker may generate a bogus token or modify the token contents (such as the authentication or attribute statements) of an existing token, causing the resource server to grant inappropriate access to the client. For example, an attacker may modify the token to extend the validity period; a malicious client may modify the assertion to gain access to information that they should not be able to view.
>
> **Token disclosure:** Tokens may contain authentication and attribute statements that include sensitive information.
>
> **Token redirect:** An attacker uses the token generated for consumption by resource server to obtain access to another resource server.
>
> **Token reuse:** An attacker attempts to use a token that has already been used once with that resource server in the past.

TOC

3.2. Threat Mitigation

A large range of threats can be mitigated by protecting the contents of the token by using a digital signature or a keyed message digest. Alternatively, the contents of the token could be passed by reference rather than by value (requiring a separate message exchange to resolve the reference to the token contents).

This document does not specify the encoding or the contents of the token; hence detailed recommendations for token integrity protection are outside the scope of this document. We assume that the token integrity protection is sufficient to prevent the token from being modified.

To deal with token redirect, it is important for the authorization server to include the identity of the intended recipients, namely a single resource server (or a list of resource servers). Restricting the use of the token to a specific scope is also recommended.

To provide protection against token disclosure, confidentiality protection is applied via TLS with a ciphersuite that offers confidentiality protection. This requires that the communication interaction between the client and the authorization server, as well as the interaction between the client and the resource server, utilize confidentiality protection. Encrypting the token contents is another alternative. Since TLS is mandatory to implement and to use with this specification, it is the preferred approach for preventing token disclosure via the communication channel. For those rare cases where the client is prevented from observing the contents of the token, token encryption has to be applied in addition to the usage of TLS protection.

To deal with token reuse, the following recommendations are made: First, the lifetime of the token has to be limited by putting a validity time field inside the protected part of the token. Note that using short-lived (one hour or less) tokens significantly reduces the impact of one of them being leaked. Second, confidentiality protection of the exchanges between the client and the authorization server and between the client and the resource server MUST be applied. As a consequence, no eavesdropper along the communication path is able to observe the token exchange. Consequently, such an on-path adversary cannot replay the token. Furthermore, the resource server MUST ensure that it only hands out tokens to clients it has authenticated first and authorized. For this purpose, the client MUST be authenticated and authorized by the resource server. The authorization server MUST also receive a confirmation (the consent of the resource owner) prior to providing a token to the client. Furthermore, when presenting the token to a resource server, the client MUST verify the identity of that resource server. Note that the client MUST validate the TLS certificate chain when making these requests to protected resources. Presenting the token to an unauthenticated and unauthorized resource server or failing to validate the certificate chain will allow adversaries to steal the token and gain unauthorized access to protected resources.

3.3. Summary of Recommendations

TOC

Safeguard bearer tokens Client implementations MUST ensure that bearer tokens are not leaked to unintended parties, as they will be able to use them to gain access to protected resources. This is the primary security consideration when using bearer tokens with OAuth and underlies all the more specific recommendations that follow.

Validate SSL certificate chains The client must validate the TLS certificate chain when making requests to protected resources. Failing to do so may enable DNS hijacking attacks to steal the token and gain unintended access.

Always use TLS (https) Clients MUST always use TLS (https) when making requests with bearer tokens. Failing to do so exposes the token to numerous attacks that could give attackers unintended access.

Don't store bearer tokens in cookies As cookies are generally sent in the clear, implementations MUST NOT store bearer tokens within them.

Issue short-lived bearer tokens Using short-lived (one hour or less) bearer tokens can reduce the impact of one of them being leaked. The User-Agent flow should only issue short lived access tokens.

Don't pass bearer tokens in page URLs Browsers may not adequately secure URLs in the browser history. If bearer tokens are passed in page URLs (typically as query string parameters), attackers might be able to steal them from the history data. Instead, pass browser tokens in message bodies for which confidentiality measures are taken.

4. IANA Considerations

TOC

This document neither establishes new IANA registries nor adds new values to existing registries.

5. References

TOC

5.1. Normative References

TOC

[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 (TXT).
[OAuth2]	Hammer-Lahav, E., Ed., Recordon, D., and D. Hardt, "The OAuth 2.0 Protocol," 2010.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[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 (TXT, PS, PDF, HTML, XML).
[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 (TXT, HTML, XML).
[RFC2818]	Rescorla, E., "HTTP Over TLS," RFC 2818, May 2000 (TXT).
[RFC3986]	Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," STD 66, RFC 3986, January 2005 (TXT, HTML, XML).
[RFC5849]	Hammer-Lahav, E., "The OAuth 1.0 Protocol," RFC 5849, April 2010 (TXT).
[W3C.REC-html401-19991224]	Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification," World Wide Web Consortium Recommendation REC-html401-19991224, December 1999 (HTML).

5.2. Informative References

TOC

[NIST800-63]	Burr, W., Dodson, D., Perlner, R., Polk, T., Gupta, S., and E. Nabbus, "NIST Special Publication 800-63-1, INFORMATION SECURITY," December 2008.

Appendix A. Acknowledgements

TOC

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: [[If your name is missing or you think someone should be added here, please send Mike Jones a note - don't be shy!]]

Michael Adams, Andrew Arnott, Dirk Balfanz, Brian Campbell, Leah Culver, Bill de hÓra, Brian Ellin, Igor Faynberg, George Fletcher, Tim Freeman, Evan Gilbert, Justin Hart, John Kemp, Eran Hammer-Lahav, Chasen Le Hara, Michael B. Jones, Torsten Lodderstedt, 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 Stübner, Paul Tarjan, and Franklin Tse.

Appendix B. Document History

TOC

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

-01

*First public draft, which incorporates feedback received on -00 including enhanced Security Considerations content. This version is intended to accompany OAuth 2.0 draft 11.

-00

*Initial draft based on preliminary version of OAuth 2.0 draft 11.

Authors' Addresses

TOC

	Michael B. Jones (editor)
	Microsoft
Email:	mbj@microsoft.com
URI:	http://self-issued.info/
	Dick Hardt
	independent

Email:	dick.hardt@gmail.com
URI:	http://dickhardt.org/
	David Recordon
	Facebook
Email:	davidrecordon@facebook.com
URI:	http://www.davidrecordon.com/