Internet Engineering Task Force (IETF)
Request for Comments: 6750
Category: Standards Track
ISSN: 2070-1721

M. Jones
Microsoft
D. Hardt
Independent

October 2012

The OAuth 2.0 Authorization Framework: Bearer Token Usage

Abstract

This specification describes how to use bearer tokens in HTTP requests to access OAuth 2.0 protected resources. Any party in possession of a bearer token (a "bearer") can use it to get access to the associated resources (without demonstrating possession of a cryptographic key). To prevent misuse, bearer tokens need to be protected from disclosure in storage and in transport.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6750.

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

OAuth enables clients to access protected resources by obtaining an access token, which is defined in "The OAuth 2.0 Authorization Framework" [RFC6749] as "a string representing an access authorization issued to the client", rather than using the resource owner's credentials directly.

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 when the OAuth access token is a bearer token.

This specification defines the use of bearer tokens over HTTP/1.1 [RFC2616] using Transport Layer Security (TLS) [RFC5246] to access protected resources. TLS is mandatory to implement and use with this specification; other specifications may extend this specification for use with other protocols. While designed for use with access tokens

resulting from OAuth 2.0 authorization [RFC6749] flows to access OAuth protected resources, this specification actually defines a general HTTP authorization method that can be used with bearer tokens from any source to access any resources protected by those bearer tokens. The Bearer authentication scheme is intended primarily for server authentication using the WWW-Authenticate and Authorization HTTP headers but does not preclude its use for proxy authentication.

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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119].

This document uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234]. Additionally, the following rules are included from HTTP/1.1 [RFC2617]: auth-param and auth-scheme; and from "Uniform Resource Identifier (URI): Generic Syntax" [RFC3986]: URI-reference.

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

1.2. Terminology

Bearer Token
: A security token with the property that any party in possession of the token (a "bearer") can use the token in any way that any other party in possession of it can. Using a bearer token does not require a bearer to prove possession of cryptographic key material (proof-of-possession).

All other terms are as defined in "The OAuth 2.0 Authorization Framework" [RFC6749].

1.3. Overview

OAuth provides a method for clients to access a protected resource on behalf of a resource owner. In the general case, before a client can access a protected resource, it must first obtain an authorization grant from the resource owner and then exchange the authorization grant for an access token. The access token represents the grant's scope, duration, and other attributes granted by the authorization grant. The client accesses the protected resource by presenting the access token to the resource server. In some cases, a client can directly present its own credentials to an authorization server to obtain an access token without having to first obtain an authorization grant from a resource owner.

The access token provides an abstraction, 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.

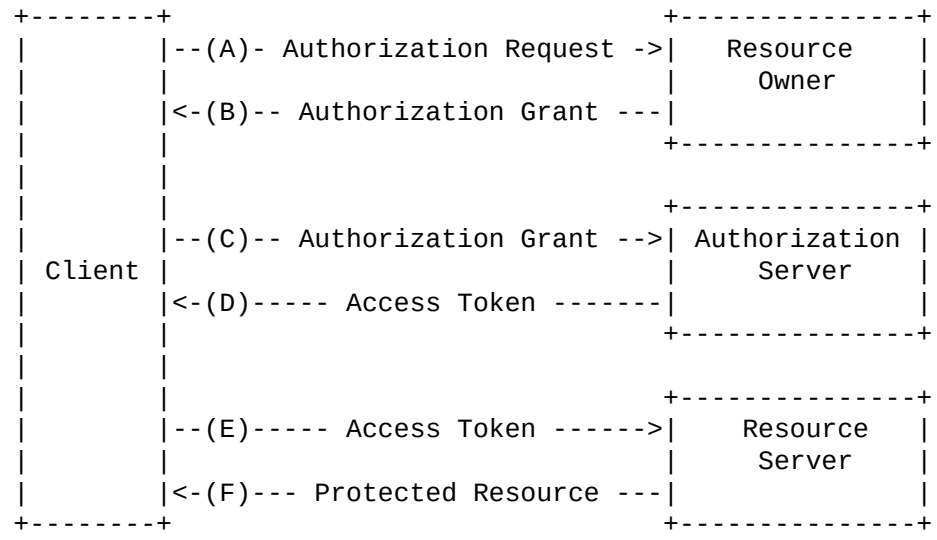

Figure 1: Abstract Protocol Flow

The abstract OAuth 2.0 flow illustrated in Figure 1 describes the interaction between the client, resource owner, authorization server, and resource server (described in [RFC6749]). The following two steps are specified within this document:

(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.

This document also imposes semantic requirements upon the access token returned in step (D).

2. Authenticated Requests

This section defines three methods of sending bearer access tokens in resource requests to resource servers. Clients MUST NOT use more than one method to transmit the token in each request.

2.1. Authorization Request Header Field

When sending the access token in the "Authorization" request header field defined by HTTP/1.1 [RFC2617], the client uses the "Bearer" authentication scheme to transmit the access token.

For example:

```
GET /resource HTTP/1.1
Host: server.example.com
Authorization: Bearer mF_9.B5f-4.1JqM
```

The syntax of the "Authorization" header field for this scheme follows the usage of the Basic scheme defined in Section 2 of [RFC2617]. Note that, as with Basic, it does not conform to the generic syntax defined in Section 1.2 of [RFC2617] but is compatible with the general authentication framework being developed for HTTP 1.1 [HTTP-AUTH], although it does not follow the preferred practice outlined therein in order to reflect existing deployments. The syntax for Bearer credentials is as follows:

```
b64token    = 1*( ALPHA / DIGIT /
                  "-" / "." / "_" / "~" / "+" / "/" ) *"="
credentials = "Bearer" 1*SP b64token
```

Clients SHOULD make authenticated requests with a bearer token using the "Authorization" request header field with the "Bearer" HTTP authorization scheme. Resource servers MUST support this method.

2.2. Form-Encoded Body Parameter

When sending the access token in the HTTP request entity-body, the client adds the access token to the request-body using the "access_token" parameter. The client MUST NOT use this method unless all of the following conditions are met:

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

- The entity-body follows the encoding requirements of the "application/x-www-form-urlencoded" content-type as defined by HTML 4.01 [W3C.REC-html401-19991224].

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

- The content to be encoded in the entity-body MUST consist entirely of ASCII [USASCII] characters.

- The HTTP request method is one for which the request-body has defined semantics. In particular, this means that the "GET" method MUST NOT be used.

The entity-body MAY include other request-specific parameters, in which case the "access_token" parameter MUST be properly separated from the request-specific parameters using "&" character(s) (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

access_token=mF_9.B5f-4.1JqM
```

The "application/x-www-form-urlencoded" method SHOULD NOT be used except in application contexts where participating browsers do not have access to the "Authorization" request header field. Resource servers MAY support this method.

2.3. URI Query Parameter

When sending the access token in the HTTP request URI, the client adds the access token to the request URI query component as defined by "Uniform Resource Identifier (URI): Generic Syntax" [RFC3986], using the "access_token" parameter.

For example, the client makes the following HTTP request using transport-layer security:

```
GET /resource?access_token=mF_9.B5f-4.1JqM HTTP/1.1
Host: server.example.com
```

The HTTP request URI query can include other request-specific parameters, in which case the "access_token" parameter MUST be properly separated from the request-specific parameters using "&" character(s) (ASCII code 38).

For example:

```
 https://server.example.com/resource?access_token=mF_9.B5f-4.1JqM&p=q
```

Clients using the URI Query Parameter method SHOULD also send a Cache-Control header containing the "no-store" option. Server success (2XX status) responses to these requests SHOULD contain a Cache-Control header with the "private" option.

Because of the security weaknesses associated with the URI method (see Section 5), including the high likelihood that the URL containing the access token will be logged, it SHOULD NOT be used unless it is impossible to transport the access token in the "Authorization" request header field or the HTTP request entity-body. Resource servers MAY support this method.

This method is included to document current use; its use is not recommended, due to its security deficiencies (see Section 5) and also because it uses a reserved query parameter name, which is counter to URI namespace best practices, per "Architecture of the World Wide Web, Volume One" [W3C.REC-webarch-20041215].

3. The WWW-Authenticate Response Header Field

If the protected resource request does not include authentication credentials or does not contain an access token that enables access to the protected resource, the resource server MUST include the HTTP "WWW-Authenticate" response header field; it MAY include it in response to other conditions as well. The "WWW-Authenticate" header field uses the framework defined by HTTP/1.1 [RFC2617].

All challenges defined by this specification MUST use the auth-scheme value "Bearer". This scheme MUST be followed by one or more auth-param values. The auth-param attributes used or defined by this specification are as follows. Other auth-param attributes MAY be used as well.

A "realm" attribute MAY be included to indicate the scope of protection in the manner described in HTTP/1.1 [RFC2617]. The "realm" attribute MUST NOT appear more than once.

The "scope" attribute is defined in Section 3.3 of [RFC6749]. The "scope" attribute is a space-delimited list of case-sensitive scope values indicating the required scope of the access token for accessing the requested resource. "scope" values are implementation defined; there is no centralized registry for them; allowed values are defined by the authorization server. The order of "scope" values is not significant. In some cases, the "scope" value will be used

when requesting a new access token with sufficient scope of access to utilize the protected resource. Use of the "scope" attribute is OPTIONAL. The "scope" attribute MUST NOT appear more than once. The "scope" value is intended for programmatic use and is not meant to be displayed to end-users.

Two example scope values follow; these are taken from the OpenID Connect [OpenID.Messages] and the Open Authentication Technology Committee (OATC) Online Multimedia Authorization Protocol [OMAP] OAuth 2.0 use cases, respectively:

```
scope="openid profile email"
scope="urn:example:channel=HBO&urn:example:rating=G,PG-13"
```

If the protected resource request included an access token and failed authentication, the resource server SHOULD include the "error" attribute to provide the client with the reason why the access request was declined. The parameter value is described in Section 3.1. In addition, the resource server MAY include the "error_description" attribute to provide developers a human-readable explanation that is not meant to be displayed to end-users. It also MAY include the "error_uri" attribute with an absolute URI identifying a human-readable web page explaining the error. The "error", "error_description", and "error_uri" attributes MUST NOT appear more than once.

Values for the "scope" attribute (specified in Appendix A.4 of [RFC6749]) MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E for representing scope values and %x20 for delimiters between scope values. Values for the "error" and "error_description" attributes (specified in Appendixes A.7 and A.8 of [RFC6749]) MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E. Values for the "error_uri" attribute (specified in Appendix A.9 of [RFC6749]) MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.

For example, in response to a protected resource request without authentication:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example"
```

And in response to a protected resource request with an authentication attempt using an expired access token:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example",
                  error="invalid_token",
                  error_description="The access token expired"
```

3.1. Error Codes

When a request fails, the resource server responds using the appropriate HTTP status code (typically, 400, 401, 403, or 405) and includes one of the following error codes in the response:

invalid_request
: The request is missing a required parameter, includes an unsupported parameter or parameter value, repeats the same parameter, uses more than one method for including an access token, or is otherwise malformed. The resource server SHOULD respond with the HTTP 400 (Bad Request) status code.

invalid_token
: The access token provided is expired, revoked, malformed, or invalid for other reasons. The resource SHOULD respond with the HTTP 401 (Unauthorized) status code. The client MAY request a new access token and retry the protected resource request.

insufficient_scope
: The request requires higher privileges than provided by the access token. The resource server SHOULD respond with the HTTP 403 (Forbidden) status code and MAY include the "scope" attribute with the scope necessary to access the protected resource.

If the request lacks any authentication information (e.g., the client was unaware that authentication is necessary or attempted using an unsupported authentication method), the resource server SHOULD NOT include an error code or other error information.

For example:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="example"
```

4. Example Access Token Response

Typically, a bearer token is returned to the client as part of an OAuth 2.0 [RFC6749] access token response. An example of such a response is:

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"mF_9.B5f-4.1JqM",
  "token_type":"Bearer",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA"
}
```

5. Security Considerations

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

5.1. Security Threats

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]. Since this document builds on the OAuth 2.0 Authorization specification [RFC6749], 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 a token generated for consumption by one resource server to gain access to a different resource server that mistakenly believes the token to be for it.

Token replay: An attacker attempts to use a token that has already been used with that resource server in the past.

5.2. Threat Mitigation

A large range of threats can be mitigated by protecting the contents of the token by using a digital signature or a Message Authentication Code (MAC). Alternatively, a bearer token can contain a reference to authorization information, rather than encoding the information directly. Such references MUST be infeasible for an attacker to guess; using a reference may require an extra interaction between a server and the token issuer to resolve the reference to the authorization information. The mechanics of such an interaction are not defined by this specification.

This document does not specify the encoding or the contents of the token; hence, detailed recommendations about the means of guaranteeing token integrity protection are outside the scope of this document. The token integrity protection MUST be 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 (the audience), typically a single resource server (or a list of resource servers), in the token. Restricting the use of the token to a specific scope is also RECOMMENDED.

The authorization server MUST implement TLS. Which version(s) ought to be implemented will vary over time and will depend on the widespread deployment and known security vulnerabilities at the time of implementation. At the time of this writing, TLS version 1.2 [RFC5246] is the most recent version, but it has very limited actual deployment and might not be readily available in implementation toolkits. TLS version 1.0 [RFC2246] is the most widely deployed version and will give the broadest interoperability.

To protect against token disclosure, confidentiality protection MUST be applied using TLS [RFC5246] with a ciphersuite that provides confidentiality and integrity 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 and integrity protection. 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 cases where the client is prevented from observing the contents of the token, token encryption MUST be applied in addition to the usage of TLS protection. As a further defense against token disclosure, the client MUST validate the TLS certificate chain when making requests to protected resources, including checking the Certificate Revocation List (CRL) [RFC5280].

Cookies are typically transmitted in the clear. Thus, any information contained in them is at risk of disclosure. Therefore, bearer tokens MUST NOT be stored in cookies that can be sent in the clear. See "HTTP State Management Mechanism" [RFC6265] for security considerations about cookies.

In some deployments, including those utilizing load balancers, the TLS connection to the resource server terminates prior to the actual server that provides the resource. This could leave the token unprotected between the front-end server where the TLS connection terminates and the back-end server that provides the resource. In such deployments, sufficient measures MUST be employed to ensure confidentiality of the token between the front-end and back-end servers; encryption of the token is one such possible measure.

To deal with token capture and replay, the following recommendations are made: First, the lifetime of the token MUST be limited; one means of achieving this is by putting a validity time field inside the protected part of the token. Note that using short-lived (one hour or less) tokens reduces the impact 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, when presenting the token to a resource server, the client MUST verify the identity of that resource server, as per Section 3.1 of "HTTP Over TLS" [RFC2818]. 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.

5.3. Summary of Recommendations

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 and underlies all the more specific recommendations that follow.

Validate TLS 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 [RFC5246] (https) or equivalent transport security 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: Implementations MUST NOT store bearer tokens within cookies that can be sent in the clear (which is the default transmission mode for cookies). Implementations that do store bearer tokens in cookies MUST take precautions against cross-site request forgery.

Issue short-lived bearer tokens: Token servers SHOULD issue short-lived (one hour or less) bearer tokens, particularly when issuing tokens to clients that run within a web browser or other environments where information leakage may occur. Using short-lived bearer tokens can reduce the impact of them being leaked.

Issue scoped bearer tokens: Token servers SHOULD issue bearer tokens that contain an audience restriction, scoping their use to the intended relying party or set of relying parties.

Don't pass bearer tokens in page URLs: Bearer tokens SHOULD NOT be passed in page URLs (for example, as query string parameters). Instead, bearer tokens SHOULD be passed in HTTP message headers or message bodies for which confidentiality measures are taken. Browsers, web servers, and other software may not adequately secure URLs in the browser history, web server logs, and other data structures. If bearer tokens are passed in page URLs, attackers might be able to steal them from the history data, logs, or other unsecured locations.

6. IANA Considerations

6.1. OAuth Access Token Type Registration

This specification registers the following access token type in the OAuth Access Token Types registry defined in [RFC6749].

6.1.1. The "Bearer" OAuth Access Token Type

Type name:
Bearer

Additional Token Endpoint Response Parameters:
(none)

HTTP Authentication Scheme(s):
Bearer

Change controller:
IETF

Specification document(s):
RFC 6750

6.2. OAuth Extensions Error Registration

This specification registers the following error values in the OAuth Extensions Error registry defined in [RFC6749].

6.2.1. The "invalid_request" Error Value

Error name:
invalid_request

Error usage location:
Resource access error response

Related protocol extension:
Bearer access token type

Change controller:
IETF

Specification document(s):
RFC 6750

6.2.2. The "invalid_token" Error Value

Error name:
invalid_token

Error usage location:
Resource access error response

Related protocol extension:
Bearer access token type

Change controller:
IETF

Specification document(s):
RFC 6750

6.2.3. The "insufficient_scope" Error Value

Error name:
insufficient_scope

Error usage location:
Resource access error response

Related protocol extension:
Bearer access token type

Change controller:
IETF

Specification document(s):
RFC 6750

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.

[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.

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

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

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[W3C.REC-html401-19991224]
Raggett, D., Le Hors, A., 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>.

[W3C.REC-webarch-20041215]
Jacobs, I. and N. Walsh, "Architecture of the World Wide Web, Volume One", World Wide Web Consortium Recommendation REC-webarch-20041215, December 2004, <http://www.w3.org/TR/2004/REC-webarch-20041215>.

7.2. Informative References

[HTTP-AUTH] Fielding, R., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", Work in Progress, October 2012.

[NIST800-63] Burr, W., Dodson, D., Newton, E., Perlner, R., Polk, T., Gupta, S., and E. Nabbus, "NIST Special Publication 800-63-1, INFORMATION SECURITY", December 2011, <http://csrc.nist.gov/publications/>.

[OMAP] Huff, J., Schlacht, D., Nadalin, A., Simmons, J., Rosenberg, P., Madsen, P., Ace, T., Rickelton-Abdi, C., and B. Boyer, "Online Multimedia Authorization Protocol: An Industry Standard for Authorized Access to Internet Multimedia Resources", April 2012, <http://www.oatc.us/Standards/Download.aspx>.

[OpenID.Messages] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., Mortimore, C., and E. Jay, "OpenID Connect Messages 1.0", June 2012, <http://openid.net/specs/openid-connect-messages-1_0.html>.

Appendix A. Acknowledgements

The following people contributed to preliminary versions of this document: Blaine Cook (BT), Brian Eaton (Google), Yaron Y. 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, the Web Resource Authorization Profiles (WRAP) community, and the OAuth Working Group. David Recordon created a preliminary version of this specification based upon an early draft of the specification that evolved into OAuth 2.0 [RFC6749]. Michael B. Jones in turn created the first version (00) of this specification using portions of David's preliminary document and edited all subsequent versions.

The OAuth Working Group has dozens of very active contributors who proposed ideas and wording for this document, including Michael Adams, Amanda Anganes, Andrew Arnott, Derek Atkins, Dirk Balfanz, John Bradley, Brian Campbell, Francisco Corella, Leah Culver, Bill de hOra, Breno de Medeiros, Brian Ellin, Stephen Farrell, Igor Faynberg, George Fletcher, Tim Freeman, Evan Gilbert, Yaron Y. Goland, Eran Hammer, Thomas Hardjono, Dick Hardt, Justin Hart, Phil Hunt, John Kemp, Chasen Le Hara, Barry Leiba, Amos Jeffries, Michael B. Jones, Torsten Lodderstedt, Paul Madsen, Eve Maler, James Manger, Laurence Miao, William J. Mills, Chuck Mortimore, Anthony Nadalin, Axel Nennker, Mark Nottingham, David Recordon, Julian Reschke, Rob Richards, Justin Richer, Peter Saint-Andre, Nat Sakimura, Rob Sayre, Marius Scurtescu, Naitik Shah, Justin Smith, Christian Stuebner, Jeremy Suriel, Doug Tangren, Paul Tarjan, Hannes Tschofenig, Franklin Tse, Sean Turner, Paul Walker, Shane Weeden, Skylar Woodward, and Zachary Zeltsan.

Authors' Addresses

Michael B. Jones
Microsoft

EMail: mbj@microsoft.com
URI: http://self-issued.info/

Dick Hardt
Independent

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