A set of SASL Mechanisms for OAuth

draft-ietf-kitten-sasl-oauth-13.txt

Abstract

OAuth enables a third-party application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction, or by allowing the third-party application to obtain access on its own behalf.

This document defines how an application client uses credentials obtained via OAuth over the Simple Authentication and Security Layer (SASL) to access a protected resource at a resource serve. Thereby, it enables schemes defined within the OAuth framework for non-HTTP-based application protocols.

Clients typically store the user’s long-term credential. This does, however, lead to significant security vulnerabilities, for example, when such a credential leaks. A significant benefit of OAuth for usage in those clients is that the password is replaced by a shared secret with higher entropy, i.e., the token. Tokens typically provide limited access rights and can be managed and revoked separately from the user’s long-term password.

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 August 18, 2014.

Copyright Notice

Copyright (c) 2014 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. Terminology ................................................. 5
3. OAuth SASL Mechanism Specifications .......................... 6
   3.1. Initial Client Response ................................. 7
   3.1.1. Reserved Key/Values ................................. 7
   3.2. Server’s Response ...................................... 8
   3.2.1. OAuth Identifiers in the SASL Context ............... 8
   3.2.2. Server Response to Failed Authentication .......... 8
   3.2.3. Completing an Error Message Sequence ............... 9
   3.3. OAuth Access Token Types using Keyed Message Digests .. 9
4. Examples .................................................. 10
   4.1. Successful Bearer Token Exchange ..................... 10
   4.2. Failed Exchange ........................................ 11
   4.3. SMTP Example of a Failed Negotiation .................. 12
5. Security Considerations ..................................... 13
6. Internationalization Considerations ........................ 14
7. IANA Considerations ......................................... 14
   7.1. SASL Registration ....................................... 14
8. References .................................................. 15
   8.1. Normative References ................................... 15
   8.2. Informative References ................................ 16
Appendix A. Acknowledgements .................................. 16
Appendix B. Document History .................................. 17
Authors’ Addresses ............................................. 19
1. Introduction

OAuth 1.0a [RFC5849] and OAuth 2.0 [RFC6749] are protocol frameworks that enable a third-party application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction, or by allowing the third-party application to obtain access on its own behalf.

The core OAuth 2.0 specification [RFC6749] specifies the interaction between the OAuth client and the authorization server; it does not define the interaction between the OAuth client and the resource server for the access to a protected resource using an Access Token. Instead, the OAuth client to resource server interaction is described in separate specifications, such as the bearer token specification [RFC6750] and the MAC Token specification [I-D.ietf-oauth-v2-http-mac]. OAuth 1.0a included the protocol specification for the communication between the OAuth client and the resource server in [RFC5849].

The main use cases for OAuth 2.0 and OAuth 1.0a have so far focused on an HTTP-based environment only. This document integrates OAuth 1.0a and OAuth 2.0 into non-HTTP-based applications using the integration into SASL. Hence, this document takes advantage of the OAuth protocol and its deployment base to provide a way to use the Simple Authentication and Security Layer (SASL) [RFC4422] to gain access to resources when using non-HTTP-based protocols, such as the Internet Message Access Protocol (IMAP) [RFC3501] and SMTP [RFC5321], which is what this memo uses in the examples.

To illustrate the impact of integrating this specification into an OAuth-enabled application environment Figure 1 shows the abstract message flow of OAuth 2.0 [RFC6749]. As indicated in the figure, this document impacts the exchange of messages (E) and (F) since SASL is used for interaction between the client and the resource server instead of HTTP.
The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable authentication mechanisms. It provides a structured interface between protocols and mechanisms. The resulting framework allows new protocols to reuse existing authentication protocols and allows old protocols to make use of new authentication mechanisms. The framework also provides a protocol for securing subsequent protocol exchanges within a data security layer.

When OAuth is integrated into SASL the high-level steps are as follows:

(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 the authorization server as an intermediary.

(B) The client receives an authorization grant which is a credential representing the resource owner’s authorization, expressed using one of four grant types defined in this specification or using an extension grant type. The authorization
grant type depends on the method used by the client to request
authorization and the types supported by the authorization server.

(C) The client requests an access token by authenticating with the
authorization server and presenting the authorization grant.

(D) The authorization server authenticates the client and
validates 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,
depends on the method used by the client to request
authorization and the types supported by the authorization server.

Again, steps (E) and (F) are not defined in [RFC6749] (but are
described in, for example, [RFC6750] for the OAuth Bearer Token
instead) and are the main functionality specified within this
document. Consequently, the message exchange shown in Figure 1 is
the result of this specification. The client will generally need to
determine the authentication endpoints (and perhaps the service
endpoints) before the OAuth 2.0 protocol exchange messages in steps
(A)-(D) are executed. The discovery of the resource owner and
authorization server endpoints is outside the scope of this
specification. The client must discover those endpoints using a
discovery mechanisms, such as Webfinger using host-meta [RFC7033].
In band discovery is not tenable if clients support the OAuth 2.0
password grant. Once credentials are obtained the client proceeds to
steps (E) and (F) defined in this specification.

OAuth 1.0 follows a similar model but uses a different terminology
and does not separate the resource server from the authorization
server.

2. Terminology

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].

The reader is assumed to be familiar with the terms used in the OAuth
2.0 specification [RFC6749].

In examples, "C:" and "S:" indicate lines sent by the client and
server respectively. Line breaks have been inserted for readability.
Note that the IMAP SASL specification requires base64 encoding, see Section 4 of [RFC4648], not this memo.

3. OAuth SASL Mechanism Specifications

SASL is used as an authentication framework in a variety of application layer protocols. This document defines the following SASL mechanisms for usage with OAuth:

**OAUTHBEARER**: OAuth 2.0 bearer tokens, as described in [RFC6750].

  RFC 6750 uses Transport Layer Security (TLS) to secure the protocol interaction between the client and the resource server.

**OAUTH10A**: OAuth 1.0a MAC tokens (using the HMAC-SHA1 keyed message digest), as described in Section 3.4.2 of [RFC5849].

New extensions may be defined to add additional OAuth Access Token Types. Such a new SASL OAuth mechanism can be added by simply registering the new name(s) and citing this specification for the further definition.

These mechanisms are client initiated and lock-step, the server always replying to a client message. In the case where the client has and correctly uses a valid token the flow is:

- Client sends a valid and correct initial client response.
- Server responds with a successful authentication.

In the case where authorization fails the server sends an error result, then client MUST then send an additional message to the server in order to allow the server to finish the exchange. Some protocols and common SASL implementations do not support both sending a SASL message and finalizing a SASL negotiation, the additional client message in the error case deals with this problem. This exchange is:

- Client sends an invalid initial client response.
- Server responds with an error message.
- Client sends a dummy client response.
- Server fails the authentication.
3.1. Initial Client Response

Client responses are a key/value pair sequence. These key/value pairs carry the equivalent values from an HTTP context in order to be able to complete an OAuth style HTTP authorization. Unknown key/value pairs MUST be ignored by the server. The ABNF [RFC5234] syntax is:

```
kvsep          = %x01
key            = 1*ALPHA
value          = *(VCHAR / SP / HTAB / CR / LF )
kvpair         = key "=" value kvsep
client_resp    = 0*kvpair kvsep
```

The following key/value pairs are defined in the client response:

- **auth (REQUIRED):** The payload of the HTTP Authorization header for an equivalent HTTP OAuth authorization.
- **host:** Contains the host name to which the client connected.
- **port:** Contains the port number represented as a decimal positive integer string without leading zeros to which the client connected.
- **qs:** The HTTP query string. This is reserved for future use, the client SHOULD NOT send it, and has the default value of ""

For OAuth token types that use keyed message digests the client MUST send host and port number key/values, and the server MUST fail an authorization request requiring keyed message digests that do not have host and port values. In OAuth 1.0a for example, the so-called "signature base string calculation" includes the reconstructed HTTP URL.

3.1.1. Reserved Key/Values

In these mechanisms values for path, query string and post body are assigned default values. OAuth authorization schemes MAY define usage of these in the SASL context and extend this specification. For OAuth Access Token Types that use request keyed message digest the default values MUST be used unless explicit values are provided in the client response. The following key values are reserved for future use:
mthd (RESERVED): HTTP method, the default value is "POST".

path (RESERVED): HTTP path data, the default value is "/".

post (RESERVED): HTTP post data, the default value is "".

3.2. Server’s Response

The server validates the response per the specification for the OAuth Access Token Types used. If the OAuth Access Token Type utilizes a keyed message digest of the request parameters then the client must provide a client response that satisfies the data requirements for the scheme in use.

The server responds to a successfully verified client message by completing the SASL negotiation. The authenticated identity reported by the SASL mechanism is the identity securely established for the client with the OAuth credential. The application, not the SASL mechanism, based on local access policy determines whether the identity reported by the mechanism is allowed access to the requested resource. Note that the semantics of the authz-id is specified by the SASL framework [RFC4422].

3.2.1. OAuth Identifiers in the SASL Context

In the OAuth framework the client may be authenticated by the authorization server and the resource owner is authenticated to the authorization server. OAuth access tokens may contain information about the authentication of the resource owner and about the client and may therefore make this information accessible to the resource server.

If both identifiers are needed by an application the developer will need to provide a way to communicate that from the SASL mechanism back to the application.

3.2.2. Server Response to Failed Authentication

For a failed authentication the server returns a JSON [RFC4627] formatted error result, and fails the authentication. The error result consists of the following values:

status (REQUIRED): The authorization error code. Valid error codes are defined in the IANA [[need registry name]] registry specified in the OAuth 2 core specification.
scope (OPTIONAL): An OAuth scope which is valid to access the
service. This may be empty which implies that unscoped tokens
are required, or a space separated list. Use of a space
separated list is NOT RECOMMENDED.

If the resource server provides a scope then the client MUST always
request scoped tokens from the token endpoint. If the resource
server provides no scope to the client then the client SHOULD presume
an empty scope (unscoped token) is needed.

If channel binding is in use and the channel binding fails the server
responds with a status code set to 412 to indicate that the channel
binding precondition failed. If the authentication scheme in use
does not include signing the server SHOULD revoke the presented
credential and the client SHOULD discard that credential.

3.2.3. Completing an Error Message Sequence

Section 3.6 of [RFC4422] explicitly prohibits additional information
in an unsuccessful authentication outcome. Therefore, the error
message is sent in a normal message. The client MUST then send an
additional client response consisting of a single %x01 (control A)
character to the server in order to allow the server to finish the
exchange.

3.3. OAuth Access Token Types using Keyed Message Digests

OAuth Access Token Types may use keyed message digests and the client
and the resource server may need to perform a cryptographic
computation for integrity protection and data origin authentication.

OAuth is designed for access to resources identified by URIs. SASL
is designed for user authentication, and has no facility for more
fine-grained access control. In this specification we require or
define default values for the data elements from an HTTP request
which allow the signature base string to be constructed properly.
The default HTTP path is "/" and the default post body is empty.
These atoms are defined as extension points so that no changes are
needed if there is a revision of SASL which supports more specific
resource authorization, e.g., IMAP access to a specific folder or FTP
access limited to a specific directory.

Using the example in the OAuth 1.0a specification as a starting
point, on an IMAP server running on port 143 and given the OAuth 1.0a
style authorization request (with %x01 shown as ^A and line breaks
added for readability) below:
n,a=user@example.com
host=example.com
user=user@example.com
port=143
auth=OAuth realm="Example",
oauth_consumer_key="9djdj82h48dj9d2",
oauth_token="k9k9d7dh3k39sjv7",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="137131201",
oauth_nonce="7d8f3e4a",
oauth_signature="Tm90IGEgcmVhbCBzaWduYXR1cmU\%3D"

The signature base string would be constructed per the OAuth 1.0 specification [RFC5849] with the following things noted:

- The method value is defaulted to POST.
- The scheme defaults to be "http", and any port number other than 80 is included.
- The path defaults to "/".
- The query string defaults to "".

In this example the signature base string with line breaks added for readability would be:

POST&http%3A%2F%2Fexample.com:143%2F&oauth_consumer_key%3D9djdj82h48dj9d2&oauth_nonce%3D7d8f3e4a&oauth_signature_method%3DHMAC-SHA1&oauth_timestamp%3D137131201&oauth_token%3Dk9k9d7dh3k39sjv7

4. Examples

These examples illustrate exchanges between an IMAP and SMTP clients and servers.

Note to implementers: The SASL OAuth method names are case insensitive. One example uses "Bearer" but that could as easily be "bearer", "BEARER", or "BeArEr".

4.1. Successful Bearer Token Exchange

This example shows a successful OAuth 2.0 bearer token exchange. Note that line breaks are inserted for readability and the underlying TLS establishment is not shown either.
As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response (with %x01 represented as ^A and long lines wrapped for readability) is:

n,a=user@example.com^Ahost=server.example.com^Aport=143^A
auth=Bearer vF9dft4qmTc2Nvb3RlckBhbHRhdmlzdGEuY29tCg==^A^A

The same credential used in an SMTP exchange is shown below. Note that line breaks are inserted for readability, and that the SMTP protocol terminates lines with CR and LF characters (ASCII values 0x0D and 0x0A), these are not displayed explicitly in the example.

[connection begins]
S: 220 mx.example.com ESMTP 12sm2095603fks.9
C: EHLO sender.example.com
S: 250-mx.example.com at your service,[172.31.135.47]
S: 250-SIZE 35651584
S: 250-8BITMIME
S: 250-AUTH LOGIN PLAIN OAUTHBEARER
S: 250-ENHANCEDSTATUSCODES
S: 250 PIPELINING
C: t1 AUTHENTICATE OAUTHBEARER bixhPXVzZXJAXhbbX BsZ5jb20BaG9zdD1zZX
   J2ZXluc2Nyb25uZXR0aG9yY29uZGl0b3IubG9ydGVkY29uc3B1bGQ4
   GMyTeziMIJUycXVzZXJUb3d3dGluZyzvUco=
S: 235 Authentication successful.
[connection continues...]

4.2. Failed Exchange

This example shows a failed exchange because of the empty Authorization header, which is how a client can query for the needed scope. Note that line breaks are inserted for readability.
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR IMAP4rev1 Server Ready
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTHBEARER cD10bHMtdW5pcXVlLGE9dXNlcckBleGFtcGxlLmNvbQFob3N0PXNlcnZlci5leGFtcGxlLmNvbQFwb3J0PTE0MwFhdXRoPQYmRhG9AQ==
S: + ewoic3RhdHVzIjoiNDAxIgoic2NvcGUiOiJleGFtcGxlX3Njb3B1Igp9
C: + AQ==
S: t1 NO SASL authentication failed

The decoded initial client response is:

n,a=user@example.com,^Ahost=server.example.com^A
port=143^Aauth=^A^A

The decoded server error response is:

{  
  "status":"401",  
  "scope":"example_scope"  
}

The client responds with the required dummy response.

4.3. SMTP Example of a Failed Negotiation

This example shows an authorization failure in an SMTP exchange. Note that line breaks are inserted for readability, and that the SMTP protocol terminates lines with CR and LF characters (ASCII values 0x0D and 0x0A), these are not displayed explicitly in the example.

[connection begins]
S: 220 mx.example.com ESMTP 12sm2095603fks.9
C: EHLO sender.example.com
S: 250-mx.example.com at your service,[172.31.135.47]
S: 250-SIZE 35651584
S: 250-8BITMIME
S: 250-AUTH LOGIN PLAIN OAUTHBEARER
S: 250-ENHANCEDSTATUSCODES
S: 250 PIPELINING
C: AUTH OAUTHBEARER bixhPT1zb2lldXIckBleGFtcGxlLmNvbQFhdXRoPUJlYXJlciB2RjlkZnQ0cW1UYzJ0dmiZUzSaGrtbHpR0V1WI5dENnPT0BAQ==
S: 334 eyJzdGF0dXMiOiI0MDEiLCJzY2hlbWVzIjoiYmVhcmVyIG1hYyIsInNjb3BlIjoiaHR0cHM6Ly9tYWlsLmdvb2dsZS5jb20vIn0K
C: AQ==
S: 535-5.7.1 Username and Password not accepted. Learn more at
S: 535 5.7.1 http://support.example.com/mail/oauth
[connection continues...]

The server returned an error message in the 334 SASL message, the client responds with the required dummy response, and the server finalizes the negotiation.

5. Security Considerations

OAuth 1.0a and OAuth 2 allows for a variety of deployment scenarios, and the security properties of these profiles vary. As shown in Figure 1 this specification is aimed to be integrated into a larger OAuth deployment. Application developers therefore need to understand the needs of their security requirements based on a threat assessment before selecting a specific SASL OAuth mechanism. For OAuth 2.0 a detailed security document [RFC6819] provides guidance to select those OAuth 2.0 components that help to mitigate threats for a given deployment. For OAuth 1.0a Section 4 of RFC 5849 [RFC5849] provides guidance specific to OAuth 1.0.

This document specifies three SASL Mechanisms for OAuth and each comes with different security properties.

OAUTHBEARER: This mechanism borrows from OAuth 2.0 bearer tokens [RFC6750]. It relies on the application using TLS to protect the OAuth 2.0 Bearer Token exchange; without TLS usage at the application layer this method is completely insecure. Consequently, TLS MUST be provided by the application when choosing this authentication mechanism.

OAUTH10A: This mechanism re-uses OAuth 1.0a MAC tokens (using the HMAC-SHA1 keyed message digest), as described in Section 3.4.2 of [RFC5849]. To compute the keyed message digest in the same way as in RFC 5839 this specification conveys additional parameters between the client and the server. This SASL mechanism only supports client authentication. If server-side authentication is desirable then it must be provided by the application underneath the SASL layer. The use of TLS is strongly RECOMMENDED.

Additionally, the following aspects are worth pointing out:

An access token is not equivalent to the user’s long term password.

Care has to be taken when these OAuth credentials are used for actions like changing passwords (as it is possible with some protocols, e.g., XMPP). The resource server should ensure that actions taken in the authenticated channel are appropriate to the strength of the presented credential.

Lifetime of the application sessions.
It is possible that SASL will be authenticating a connection and the life of that connection may outlast the life of the access token used to establish it. This is a common problem in application protocols where connections are long-lived, and not a problem with this mechanism per se. Resource servers may unilaterally disconnect clients in accordance with the application protocol.

Access tokens have a lifetime.

Reducing the lifetime of an access token provides security benefits and OAuth 2.0 introduces refresh tokens to obtain new access token on the fly without any need for a human interaction. Additionally, a previously obtained access token may be revoked or rendered invalid at any time by the authorization server. The client may request a new access token for each connection to a resource server, but it should cache and re-use valid credentials.

6. Internationalization Considerations

The identifier asserted by the OAuth authorization server about the resource owner inside the access token may be displayed to a human. For example, when SASL is used in the context of IMAP the resource server may assert the resource owner’s email address to the IMAP server for usage in an email-based application. The identifier may therefore contain internationalized characters and an application needs to ensure that the mapping between the identifier provided by OAuth is suitable for use with the application layer protocol SASL is incorporated into.

At the time of writing the standardization of the various claims in the access token (in JSON format) is still ongoing, see [I-D.ietf-oauth-json-web-token]. Once completed it will provide a standardized format for exchanging identity information between the authorization server and the resource server.

7. IANA Considerations

7.1. SASL Registration

The IANA is requested to register the following SASL profile:

SASL mechanism profile: OAUTHBEARER

Security Considerations: See this document

Published Specification: See this document
For further information: Contact the authors of this document.

Owner/Change controller: the IETF

Note: None

The IANA is requested to register the following SASL profile:

SASL mechanism profile: OAUTH10A

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Owner/Change controller: the IETF

Note: None

8. References

8.1. Normative References


8.2. Informative References


Appendix A. Acknowlegements

The authors would like to thank the members of the Kitten working group, and in addition and specifically: Simon Josefson, Torsten Lodderstadt, Ryan Troll, Alexey Melnikov, Jeffrey Hutzelman, and Nico Williams.

This document was produced under the chairmanship of Alexey Melnikov, Tom Yu, Shawn Emery, Josh Howlett, Sam Hartman. The supervising area directors was Stephen Farrell.
Appendix B.  Document History

-13
  o Changed affiliation.

-12
  o Removed -PLUS components from the specification.

-11
  o Removed GSS-API components from the specification.
  o Updated security consideration section.

-10
  o Clarifications throughout the document in response to the feedback from Jeffrey Hutzelman.

-09
  o Incorporated review by Alexey and Hannes.
  o Clarified the three OAuth SASL mechanisms.
  o Updated references
  o Extended acknowledgements

-08
  o Fixed the channel binding examples for p=$cbtype
  o More tuning of the authcid language and edited and renamed 3.2.1.

-07
  o Struck the MUST language from authzid.

-06
  o Removed the user field. Fixed the examples again.
o Added canonicalization language.

-05

o Fixed the GS2 header language again.

o Separated out different OAuth schemes into different SASL mechanisms. Took out the scheme in the error return. Tuned up the IANA registrations.

o Added the user field back into the SASL message.

o Fixed the examples (again).

-04

o Changed user field to be carried in the gs2-header, and made gs2 header explicit in all cases.

o Converted MAC examples to OAuth 1.0a. Moved MAC to an informative reference.

o Changed to sending an empty client response (single control-A) as the second message of a failed sequence.

o Fixed channel binding prose to refer to the normative specs and removed the hashing of large channel binding data, which brought more problems than it solved.

o Added a SMTP examples for Bearer use case.

-03

o Added user field into examples and fixed egregious errors there as well.

o Added text reminding developers that Authorization scheme names are case insensitive.

-02

o Added the user data element back in.

o Minor editorial changes.
-01

- Ripping out discovery. Changed to refer to I-D.jones-appsawg-webfinger instead of WF and SWD older drafts.

- Replacing HTTP as the message format and adjusted all examples.

-00

- Renamed draft into proper IETF naming format now that it’s adopted.

- Minor fixes.

Authors’ Addresses

William Mills
Yahoo! Inc.

Email: wmills_92105@yahoo.com

Tim Showalter

Email: tjs@psaux.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge  CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at