KITTEN
Internet-Draft
Intended status: Standards Track
Expires: March 20, 2015

W. Mills
Microsoft
T. Showalter

H. Tschofenig
ARM Ltd.

September 16, 2014

A set of SASL Mechanisms for OAuth
draft-ietf-kitten-sasl-oauth-16.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 March 20, 2015.

Copyright Notice

Table of Contents

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 [RFC2616] 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 the Simple Mail Transfer Protocol (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.

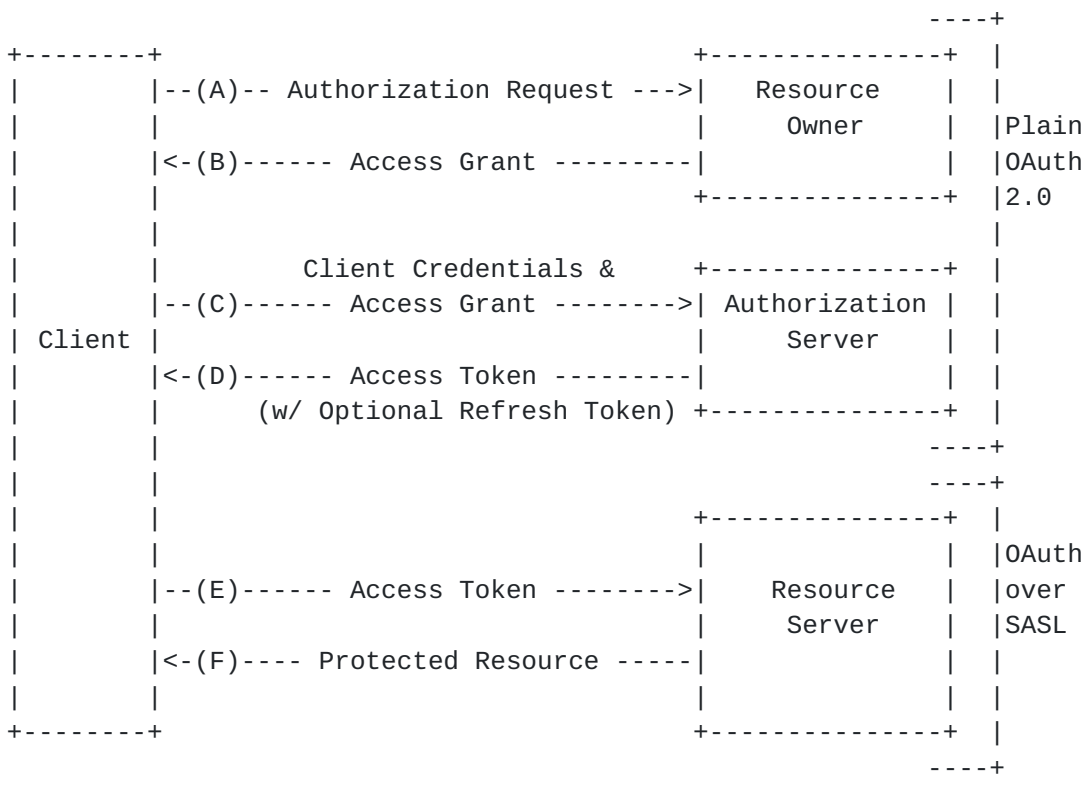

Figure 1: OAuth 2.0 Protocol Flow

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 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 the grant types defined in [RFC6749] or [RFC5849] 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, indicates a successful authentication.

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 mechanism, 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] and SASL [RFC4422].

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:

1. Client sends a valid and correct initial client response.

2. 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:

1. Client sends an invalid initial client response.

2. Server responds with an error message.

3. Client sends a dummy client response.

4. Server fails the authentication.

3.1. Initial Client Response

Client responses are a GS2 [RFC5801] header followed by zero or more key/value pairs, or may be empty. The gs2-header is defined here for compatibility with GS2 if a GS2 mechanism is formally defined, but this document does not define one. These key/value pairs take the place of the corresponding HTTP headers and values to convey the information necessary 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
;;gs2-header     = See RFC 5801
  client_resp    = (gs2-header kvsep 0*kvpair kvsep) / kvsep
```

The GS2 header MAY include the user name associated with the resource being accessed, the "authzid". It is worth noting that application protocols are allowed to require an authzid, as are specific server implementations.

The following keys and corresponding values are defined in the client response:

auth (REQUIRED): The payload that would be in the HTTP Authorization header if this OAuth exchange was being carried out over HTTP.

host: Contains the host name to which the client connected, in an HTTP context this is the value of the HTTP Host header.

port: Contains the port number represented as a decimal positive integer string without leading zeros to which the client connected.

For OAuth token types such as OAuth 1.0a 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 are not accompanied by 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 "".

qs (RESERVED): The HTTP query string, the default value is "".

3.2. Server's Response

The server validates the response according 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 "OAuth Extensions Error 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 scope value. If a scope is specified then a single scope is preferred, use of a space separated list of scopes is NOT RECOMMENDED.

oauth-configuration (OPTIONAL): The URL for for a document following the OpenID Provider Configuration Information schema as described in OpenID Connect Discovery [OpenID.Discovery] section 3 that is appropriate for the user. This document MUST have all OAuth related data elements populated. The server MAY return different URLs for users in different domains and the client SHOULD NOT cache a single returned value and assume it applies for all users/domains that the server suports.

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 required to access the resource.

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,^A
host=example.com^A
port=143^A
auth=OAuth realm="Example",
           oauth_consumer_key="9djdj82h48djs9d2",
           oauth_token="kkk9d7dh3k39sjv7",
           oauth_signature_method="HMAC-SHA1",
           oauth_timestamp="137131201",
           oauth_nonce="7d8f3e4a",
           oauth_signature="Tm90IGEgcmVhbCBzaWduYXR1cmU"^A^A
```

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%3D9djdj82h4
8djs9d2%26oauth_nonce%3D7d8f3e4a%26oauth_signature_method%3DHMAC-SH
A1%26oauth_timestamp%3D137131201%26oauth_token%3Dkkk9d7dh3k39sjv7

4. Examples

These examples illustrate exchanges between 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 in IMAP. Note that line breaks are inserted for readability and the underlying TLS establishment is not shown either.

```
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAWhvc3Q9c2
      VydmVyLmV4YW1wbGUuY29tAXBvcnQ9MTQzAWF1dGg9QmVhcmVyIHZGOWRmdDRxb
      VRjMk52YjNSbGNrQmhiSFJoZG1semRHRXVZMjl0Q2c9PQEB
S: t1 OK SASL authentication succeeded
```

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 bixhPXVzZXJAZXhhbXBsZS5jb20sAWhvc3Q9c
      2VydmVyLmV4YW1wbGUuY29tAXBvcnQ9MTQzAWF1dGg9QmVhcmVyIHZGOWRmdDR
      xbVRjMk52YjNSbGNrQmhiSFJoZG1semRHRXVZMjl0Q2c9PQEB
S: 235 Authentication successful.
[connection continues...]
```

4.2. Successful OAuth 1.0a Token Exchange

This IMAP example shows a successful OAuth 1.0a token exchange. Note that line breaks are inserted for readability and the underlying TLS establishment is not shown. Signature computation is discussed in Section 3.3.

```
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER OAUTH10A SASL-IR
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH10A bixhPXVzZXJAZXhhbXBsZS5jb20sAWhvc3Q9ZXhhb
      XBsZS5jb20BcG9ydD0xNDMBYXV0aD1PQXV0aCByZWFsbT0iRXhhbXBsZSIsb2F1
      dGhfY29uc3VtZXJfa2V5PSI5ZGpkajgyaDQ4ZGpzOWQyIixvYXV0aF90b2tlbj0
      ia2trOWQ3ZGgzazM5c2p2NyIsb2F1dGhfc2lnbmF0dXJlX21ldGhvZD0iSE1BQy
      1TSEExIixvYXV0aF90aW1lc3RhbXA9IjEzNzEzMTIwMSIsb2F1dGhfbm9uY2U9I
      jdkOGYzZTRhIixvYXV0aF9zaWduYXR1cmU9IlRtOTBJR0VnY21WaGJDQnphV2R1
      WVhSMWNtVSUzRCIBAQ==
S: t1 OK SASL authentication succeeded
```

As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response (with %x01 represented as ^A and lines wrapped for readability) is:

```
   n,a=user@example.com,^A
   host=example.com^A
   port=143^A
   auth=OAuth realm="Example",
              oauth_consumer_key="9djdj82h48djs9d2",
              oauth_token="kkk9d7dh3k39sjv7",
              oauth_signature_method="HMAC-SHA1",
              oauth_timestamp="137131201",
              oauth_nonce="7d8f3e4a",
              oauth_signature="SSdtIGEgbGl0dGxlIHRlYSBwb3Qu"^A^A
```

4.3. Failed Exchange

This IMAP 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: * OK IMAP4rev1 Server Ready
   C: t0 CAPABILITY
   S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR IMAP4rev1 Server
        Ready
   S: t0 OK Completed
   C: t1 AUTHENTICATE OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAW
         hvc3Q9c2VydmVyLmV4YW1wbGUuY29tAXBvcnQ9MTQzAWF1dGg9AQE=
   S: + eyJzdGF0dXMiOiJpbnZhbGlkX3Rva2VuIiwic2NvcGUiOiJleGFtcGxl
        X3Njb3BlIiwib3BlbmlkLWNvbmZpZ3VyYXRpb24iOiJodHRwczovL2V4
        YW1wbGUuY29tLy53ZWxsLWtub3duL29wZW5pZC1jb25maWd1cmF0aW9u
        In0=
   S: + eyJzdGF0dXMiOiI0MDEiLCJzY29wZSI6ImV4YW1wbGVfc2NvcGUiLCJv
        cGVuaWQtY29uZmlndXJhdGlvbiI6Imh0dHBzOi8vZXhhbXBsZS5jb20v
        LndlbGwta25vd24vb3BlbmlkLWNvbmZpZ3VyYXRpb24ifQ==
   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":"invalid_token",
"scope":"example_scope",
"openid-configuration":"https://example.com/.well-known/openid-configuration"
}
```

The client responds with the required dummy response, "AQ==" is the base64 encoding of the ASCII value 0x01.

4.4. 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 bix1c2VyPXNvbWV1c2VyQGV4YW1wbGUuY29tLAFhdXRoPUJlYXJl
       ciB2RjlkZnQ0cW1UYzJOdmIzUmxja0JoZEhSaGRtbHpkR0V1WTI5dENnPT0BAQ==
S: 334 eyJzdGF0dXMiOiI0MDEiLCJzY2hlbWVzIjoiYmVhcmVyIG1hYyIsInNjb3BlIjoia
       HR0cHM6Ly9tYWlsLmdvb2dsZS5jb20vIn0K
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 two 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 was 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 desireable 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 [RFC6120]). The resource server should ensure that actions taken in the authenticated channel are appropriate to the strength of the presented credential.

Lifetime of the appliation 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 might be revoked or rendered invalid at any time. 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 identifer 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 client 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

[OpenID.Discovery] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", July 2011.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3174] Eastlake, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[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.

[RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, July 2010.

[RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.

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

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.

8.2. Informative References

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-25 (work in progress), July 2014.

[I-D.ietf-oauth-v2-http-mac]
Richer, J., Mills, W., Tschofenig, H., and P. Hunt, "OAuth 2.0 Message Authentication Code (MAC) Tokens", draft-ietf-oauth-v2-http-mac-05 (work in progress), January 2014.

[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.

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, January 2013.

[RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, September 2013.

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, Nico Williams, Matt Miller, and Benjamin Kaduk.

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

Appendix B. Document History

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

- Last call feedback again. Primarily editorial changes. Corrected examples.

-15

- Last call feedack on the GS2 stuff being ripped out completely.

- Removed the "user" parameter and put stuff back into the gs2-header. Call out that the authzid goes in the gs2-header with some prose about when it might be required. Very comparable to -10.

- Added an OAuth 1.0A example explicitly.

-14

- Last call feedack on RFC citations needed, small editorial.

- Added the "user" parameter back, which was pulled when we started down the GS2 path. Same language as -03.

- Defined a stub GS2 header to make sure that when the GS2 bride is defined for this that nothing will break when it actually starts to get populated.

-13

- Changed affiliation.

-12

- Removed -PLUS components from the specification.

-11

- Removed GSS-API components from the specification.

- Updated security consideration section.

-10

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

-09

- Incorporated review by Alexey and Hannes.

- Clarified the three OAuth SASL mechanisms.
- Updated references
- Extended acknowledgements

-08

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

-07

- Struck the MUST langiage from authzid.
-

-06

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

-05

- Fixed the GS2 header language again.
- Separated out different OAuth schemes into different SASL mechanisms. Took out the scheme in the error return. Tuned up the IANA registrations.
- Added the user field back into the SASL message.
- Fixed the examples (again).
-

-04

- Changed user field to be carried in the gs2-header, and made gs2 header explicit in all cases.
- Converted MAC examples to OAuth 1.0a. Moved MAC to an informative reference.

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

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

- Added a SMTP examples for Bearer use case.

-03

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

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

-02

- Added the user data element back in.

- 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
Microsoft

Email: wimills@microsoft.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