Internet Engineering Task Force (IETF)
Request for Comments: 7628
Category: Standards Track
ISSN: 2070-1721

W. Mills
Microsoft
T. Showalter

H. Tschofenig
ARM Ltd.

August 2015

A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth

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 server. 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 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/rfc7628.

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 by orchestrating an approval interaction on behalf of a resource owner 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]. OAuth 1.0a includes the protocol specification for the communication between the OAuth client and the resource server in [RFC5849].

The main use cases for OAuth 1.0a and OAuth 2.0 have so far focused on an HTTP-based [RFC7230] environment only. This document integrates OAuth 1.0a and OAuth 2.0 into non-HTTP-based applications using the integration into the Simple Authentication and Security Layer (SASL) [RFC4422]. Hence, this document takes advantage of the OAuth protocol and its deployment base to provide a way to use SASL 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]. This document gives examples of use in IMAP and SMTP.

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.

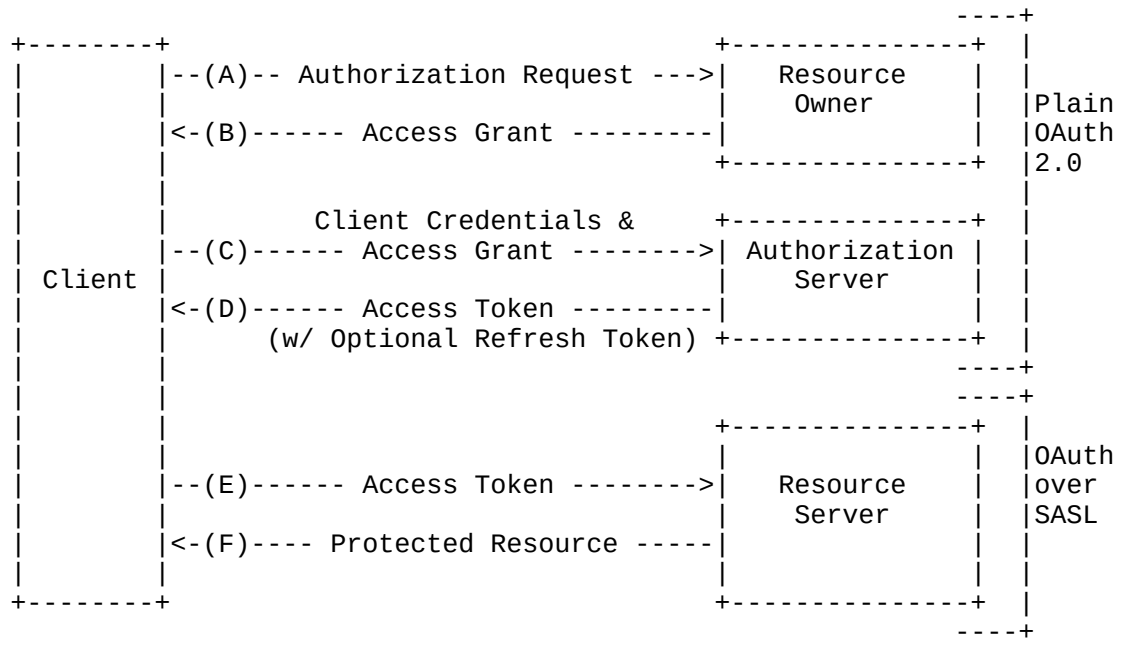

Figure 1: OAuth 2.0 Protocol Flow

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 mechanisms 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 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, it issues an access token.

(E) The client requests the protected resource from the resource server and authenticates it by presenting the access token.

(F) The resource server validates the access token, and if valid, it 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, authorization server endpoints, and client registration are outside the scope of this specification. The client must discover the authorization endpoints using a discovery mechanism such as OpenID Connect Discovery (OIDCD) [OpenID.Discovery] or WebFinger using host-meta [RFC7033]. Once credentials are obtained, the client proceeds to steps (E) and (F) defined in this specification. Authorization endpoints MAY require client registration, and generic clients SHOULD support the Dynamic Client Registration protocol [RFC7591].

OAuth 1.0a 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", "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, as specified in Section 4 of [RFC4648].

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) [RFC5246] to secure the protocol interaction between the client and the resource server.

OAUTH10A: OAuth 1.0a Message Authentication Code (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 registering the new name(s) with IANA in the SASL Mechanisms registry and citing this specification for the further definition.

SASL mechanisms using this document as their definition do not provide a data security layer; that is, they cannot provide integrity or confidentiality protection for application messages after the initial authentication. If such protection is needed, TLS or some similar solution should be used. Additionally, for the two mechanisms specified in this document, TLS MUST be used for OAUTHBEARER to protect the bearer token; for OAUTH10A, the use of TLS is RECOMMENDED.

These mechanisms are client initiated and in lockstep, with 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 authentication fails, the server sends an error result; the 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 it may be empty. The gs2-header rule is defined here as a placeholder for compatibility with GS2 if a GS2 mechanism is formally defined, but this document does not define one. The 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 *kvpair kvsep) / kvsep
```

The GS2 header MAY include the username 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 client response consisting of only a single kvsep is used only when authentication fails and is only valid in that context. If sent as the first message from the client, the server MAY simply fail the authentication without returning discovery information since there is no user or server name indication.

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 hostname to which the client connected. In an HTTP context, this is the value of the HTTP Host header.

port: Contains the destination port that the client connected to, represented as a decimal positive integer string without leading zeros.

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 include a keyed message digest of the request, 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 the empty string ("").

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

3.2. Server’s Response

The server validates the response according to 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 fully validates the client response before generating a server response; this will necessarily include the validation steps listed in the specification for the OAuth Access Token Type used. However, additional validation steps may be needed, depending on the particular application protocol making use of SASL. In particular, values included as kvpairs in the client response (such as host and

port) that correspond to values known to the application server by some other mechanism (such as an application protocol data unit or preconfigured values) MUST be validated to match between the initial client response and the other source(s) of such information. As a concrete example, when SASL is used over IMAP to an IMAP server for a single domain, the hostname can be available via configuration; this hostname must be validated to match the value sent in the 'host' kvpair.

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 authzid are 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 an error result in JSON [RFC7159] format 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" as specified in the OAuth 2.0 core specification.

scope (OPTIONAL): An OAuth scope that is valid to access the service. This may be omitted, which implies that unscoped tokens are required. If a scope is specified, then a single scope is preferred. At the time this document was written, there are several implementations that do not properly support space-separated lists of scopes, so the use of a space-separated list of scopes is NOT RECOMMENDED.

openid-configuration (OPTIONAL): The URL for a document following the OpenID Provider Configuration Information schema as described in OIDCD [OpenID.Discovery], Section 3 that is appropriate for the user. As specified in OIDCD, this will have the "https" URL scheme. 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 supports. The returned discovery document SHOULD have all data elements required by the OpenID Connect Discovery specification populated. In addition, the discovery document SHOULD contain the 'registration_endpoint' element to identify the endpoint to be used with the Dynamic Client Registration protocol [RFC7591] to obtain the minimum number of parameters necessary for the OAuth protocol exchange to function. Another comparable discovery or client registration mechanism MAY be used if available.

The use of the 'offline_access' scope, as defined in [OpenID.Core], is RECOMMENDED to give clients the capability to explicitly request a refresh token.

If the resource server provides a scope, then the client MUST always request scoped tokens from the token endpoint. If the resource server does not return a scope, the client SHOULD presume an unscoped token is required to access the resource.

Since clients may interact with a number of application servers, such as email servers and Extensible Messaging and Presence Protocol (XMPP) [RFC6120] servers, they need to have a way to determine whether dynamic client registration has been performed already and whether an already available refresh token can be reused to obtain an access token for the desired resource server. This specification RECOMMENDS that a client uses the information in the 'iss' element defined in OpenID Connect Core [OpenID.Core] to make this determination.

3.2.3. Completing an Error Message Sequence

Section 3.6 of SASL [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 either 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 or a SASL abort message as generally defined in Section 3.5 of SASL [RFC4422]. A specific example of an abort message is the "BAD" response to an AUTHENTICATE in IMAP [RFC3501], Section 6.2.2.

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 that allows 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 that 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, below is the authorization request in OAuth 1.0a style (with %x01 shown as ^A and line breaks added for readability), assuming it is on an IMAP server running on port 143:

```
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 1.0a 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. All IMAP examples use SASL-IR [RFC4959] and send payload in the initial client response. The bearer token examples assume encrypted transport; if the underlying connection is not already TLS, then STARTTLS MUST be used as TLS is required in the bearer token specification.

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.

```
[Initial connection and TLS establishment...]
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAWhv
      c3Q9c2VydmVyLmV4YW1wbGUuY29tAXBvcnQ9MTQzAWF1dGg9QmVhcmVyI
      HZGOWRmdDRxbVRjMk52YjNSbGNrQmhiSFJoZG1semRHRXVZMjl0Q2c9PQ
      EB
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. Again, this example assumes that TLS is already established per the bearer token specification requirements.

```
[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-STARTTLS
S: 250 PIPELINING
[Negotiate TLS...]
C: t1 AUTH OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAWhvc3Q9c2Vy
      dmVyLmV4YW1wbGUuY29tAXBvcnQ9NTg3AWF1dGg9QmVhcmVyIHZGOWRmd
      DRxbVRjMk52YjNSbGNrQmhiSFJoZG1semRHRXVZMjl0Q2c9PQEB
S: 235 Authentication successful.
[connection continues...]
```

The decoded initial client response is:

```
n,a=user@example.com,^Ahost=server.example.com^Aport=587^A
auth=Bearer vF9dft4qmTc2Nvb3RlckBhbHRhdmlzdGEuY29tCg==^A^A
```

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. This example assumes that TLS is already established. Signature computation is discussed in Section 3.3.

```
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER AUTH=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
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAW
      hvc3Q9c2VydmVyLmV4YW1wbGUuY29tAXBvcnQ9MTQzAWF1dGg9AQE=
S: + eyJzdGF0dXMiOiJpbnZhbGlkX3Rva2VuIiwic2NvcGUiOiJleGFtcGxl
     X3Njb3BlIiwib3BlbmlkLWNvbmZpZ3VyYXRpb24iOiJodHRwczovL2V4
     YW1wbGUuY29tLy53ZWxsLWtub3duL29wZW5pZC1jb25maWd1cmF0aW9u
     In0=
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-config"
}
```

The client responds with the required dummy response; "AQ==" is the base64 encoding of the ASCII value 0x01. The same exchange using the IMAP-specific method of canceling an AUTHENTICATE command sends "*" and is shown below.

```
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR IMAP4rev1
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAW
     hvc3Q9c2VydmVyLmV4YW1wbGUuY29tAXBvcnQ9MTQzAWF1dGg9AQE=
S: + eyJzdGF0dXMiOiJpbnZhbGlkX3Rva2VuIiwic2NvcGUiOiJleGFtcGxl
     X3Njb3BlIiwib3BlbmlkLWNvbmZpZ3VyYXRpb24iOiJodHRwczovL2V4
     YW1wbGUuY29tLy53ZWxsLWtub3duL29wZW5pZC1jb25maWd1cmF0aW9u
     In0=
C: *
S: t1 NO SASL authentication failed
```

4.4. SMTP Example of a Failed Negotiation

This example shows an authorization failure in an SMTP exchange. TLS negotiation is not shown, but as noted above, it is required for the use of bearer tokens.

```
[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
[Negotiate TLS...]
C: AUTH OAUTHBEARER bix1c2VyPXNvbWV1c2VyQGV4YW1wbGUuY29tLAFhdXRoPUJlYXJl
     ciB2RjlkZnQ0cW1UYzJOdmIzUmxja0JoZEhSaGRtbHpkR0V1WTI5dENnPT0BAQ==
S: 334 eyJzdGF0dXMiOiJpbnZhbGlkX3Rva2VuIiwic2NoZW1lcyI6ImJlYXJlciBtYWMiL
     CJzY29wZSI6Imh0dHBzOi8vbWFpbC5leGFtcGxlLmNvbS8ifQ==
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 initial client response is:

```
n,user=someuser@example.com,^A
auth=Bearer vF9dft4qmTc2Nvb3RlckBhdHRhdmlzdGEuY29tCg==^A^A
```

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.

```
{
    "status":"invalid_token",
    "schemes":"bearer mac",
    "scope":"https://mail.example.com/"
}
```

5. Security Considerations

OAuth 1.0a and OAuth 2.0 allow 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 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 [RFC5849] provides guidance specific to OAuth 1.0a.

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 reuses 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 [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 application sessions.

It is possible that SASL will be used to authenticate 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 tokens on the fly without any need for 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 reuse 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 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. An example of a SASL-compatible container is the JSON Web Token (JWT) [RFC7519], which provides a standardized format for exchanging authorization and identity information that supports internationalized characters.

7. IANA Considerations

7.1. SASL Registration

The IANA has registered the following entry in the SASL Mechanisms registry:

SASL mechanism name: OAUTHBEARER

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Intended usage: COMMON

Owner/Change controller: the IESG

Note: None

The IANA has registered the following entry in the SASL Mechanisms registry:

SASL mechanism name: OAUTH10A

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Intended usage: COMMON

Owner/Change controller: the IESG

Note: None

8. References

8.1. Normative References

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Discovery]
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", November 2014, <http://openid.net/specs/openid-connect-discovery-1_0.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <http://www.rfc-editor.org/info/rfc4422>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[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, DOI 10.17487/RFC5801, July 2010, <http://www.rfc-editor.org/info/rfc5801>.

[RFC5849] Hammer-Lahav, E., Ed., "The OAuth 1.0 Protocol", RFC 5849, DOI 10.17487/RFC5849, April 2010, <http://www.rfc-editor.org/info/rfc5849>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <http://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", RFC 7591, DOI 10.17487/RFC7591, July 2015, <http://www.rfc-editor.org/info/rfc7591>.

8.2. Informative References

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <http://www.rfc-editor.org/info/rfc3501>.

[RFC4959] Siemborski, R. and A. Gulbrandsen, "IMAP Extension for Simple Authentication and Security Layer (SASL) Initial Client Response", RFC 4959, DOI 10.17487/RFC4959, September 2007, <http://www.rfc-editor.org/info/rfc4959>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <http://www.rfc-editor.org/info/rfc5321>.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <http://www.rfc-editor.org/info/rfc6120>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <http://www.rfc-editor.org/info/rfc6819>.

[RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, DOI 10.17487/RFC7033, September 2013, <http://www.rfc-editor.org/info/rfc7033>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <http://www.rfc-editor.org/info/rfc7519>.

Acknowledgements

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, Matthew Miller, and Benjamin Kaduk. The supervising Area Director was Stephen Farrell.

Authors' Addresses

William Mills
Microsoft

Email: wmills_92105@yahoo.com

Tim Showalter

Email: tjs@psaux.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
United Kingdom

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