KITTEN	W. Mills
Internet-Draft	T. Showalter
Intended status: Standards Track	Yahoo! Inc.
Expires: May 03, 2012	H. Tschofenig
	Nokia Siemens Networks
	October 31, 2011

A SASL and GSS-API Mechanism for OAuth
draft-mills-kitten-sasl-oauth-04.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 OAuth over the Simple Authentication and Security Layer (SASL) or the Generic Security Service Application Program Interface (GSS-API) to access a protected resource at a resource serve, and additionally defines authorization and token issuing endpoint discovery. 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 token. Tokens typically provided limited access rights and can be managed and revoked separately from the user's long-term credential (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 May 03, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

OAuth [I-D.ietf-oauth-v2] 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. The core OAuth specification [I-D.ietf-oauth-v2] does not define the interaction between the client and the resource server with the access to a protected resource using an Access Token. This functionality is described in two separate specifications, namely [I-D.ietf-oauth-v2-bearer], and [I-D.ietf-oauth-v2-http-mac], whereby the focus is on an HTTP-based environment only.
Figure 1 shows the abstract message flow as shown in Figure 1 of [I-D.ietf-oauth-v2].

```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

This document takes advantage of the OAuth protocol and its deployment base to provide a way to use SASL [RFC4422] as well as the GSS-API [RFC2743] to gain access to resources when using non-HTTP-based protocols, such as the Internet Message Access Protocol (IMAP) [RFC3501], which is what this memo uses in the examples.

The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable mechanisms. It provides a structured interface between protocols and mechanisms. The resulting framework allows new protocols to reuse existing mechanisms and allows old protocols to make use of new mechanisms. The framework also provides a protocol for securing subsequent protocol exchanges within a data security layer.

The Generic Security Service Application Program Interface (GSS-API) [RFC2743] provides a framework for applications to support multiple authentication mechanisms through a unified interface.

This document defines a SASL mechanism for OAuth, but it conforms to the new bridge between SASL and the GSS-API called GS2 [RFC5801]. This means that this document defines both a SASL mechanism and a GSS-API mechanism. Implementers may be interested in either the SASL, the GSS-API, or even both mechanisms. To faciliate these two variants, the description has been split into two parts, one part that provides normative references for those interested in the SASL OAuth mechanism (see Section 3), and a second part for those implementers that wish to implement the GSS-API portion (see Section 4).

When OAuth is integrated into SASL and the GSS-API 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, serves the request.

Steps (E) and (F) are not defined in [I-D.ietf-oauth-v2] and are the main functionality specified within this document. Additionally, an optional discovery exchange is defined. Consequently, the message exchange shown in Figure 2 is the result of this specification. (1) and (2) denote the optional discovery exchange steps that may happen before the OAuth 2.0 protocol exchange messages in steps (A)-(D) are executed. Steps (E) and (F) also defined in this specification.

```
                                                              ----+
   +--------+                                  +---------------+  |
   |        |--(A)-- Authorization Request --->|   Resource    |  |
   |        |                                  |    Owner      |  |Plain
   |        |<-(B)------ Access Grant ---------|               |  |OAuth
   |        |                                  +---------------+  |2.0
   |        |                                                     |
   |        |         Client Credentials &     +---------------+  |
   |        |--(C)------ Access Grant -------->| Authorization |  |
   | Client |                                  |     Server    |  |
   |        |<-(D)------ Access Token ---------|               |  |
   |        |      (w/ Optional Refresh Token) +---------------+  |
   |        |                                                 ----+
   |        |
   |        |                                                 ----+
   |        |        (Optional discovery)      +---------------+  |
   |        |--(1)------- User Name  --------->|               |  |
   | Client |                                  |               |  |
   |        |<-(2)------ Authentication -------|               |  |
   |        |          endpoint information    |    Resource   |  |OAuth
   |        |                                  |     Server    |  |over
   |        |--(E)------ Access Token -------->|               |  |SASL/
   |        |                                  |               |  |GSS-
   |        |<-(F)---- Protected Resource -----|               |  |API
   +--------+                                  +---------------+  |
                                                              ----+
```

*Note: The discovery procedure in OAuth is still work in progress. Hence, the discovery components described in this document should be considered incomplete and a tentative proposal. In general, there is a trade off between a generic, externally available defined discovery mechanisms (such as Webfinger using host-meta [I-D.hammer-hostmeta], or [I-D.jones-simple-web-discovery]) and configuration information exchanged in-band between the SASL communication endpoints.

It is worthwhile to note that this specification is also compatible with OAuth 1.0a [RFC5849].

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 [I-D.ietf-oauth-v2].
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 message, not this memo.

3. OAuth SASL Mechanism Specification

SASL is used as a generalized authentication method in a variety of application layer protocols. This document defines two SASL mechanisms for usage with OAuth: "OAUTH" and "OAUTH-PLUS". The "OAUTH" SASL mechanism provides bearer token alike semantic for SASL while "OAUTH-PLUS" provides a semantic similar to OAuth MAC authentication by utilizing a channel binding mechanism [RFC5056].

3.1. Channel Binding

If the specification for the underlying authorization scheme requires a security layer, such as TLS [RFC5246], the server SHOULD only offer a mechanism where channel binding can be enabled.
The channel binding data is computed by the client based on it's choice of preferred channel binding type. As specified in [RFC5056], the channel binding information MUST start with the channel binding unique prefix, followed by a colon (ASCII 0x3A), followed by a base64 encoded channel binding payload. The channel binding payload is the raw data from the channel binding type if the raw channel binding data is less than 500 bytes. If the raw channel binding data is 500 bytes or larger then a SHA-1 [RFC3174] hash of the raw channel binding data is computed.
If the client is using tls-unique for a channel binding then the raw channel binding data equals the first TLS finished message. This is under the 500 byte limit, so the channel binding payload sent to the server would be the base64 encoded first TLS finished message.
In the case where the client has chosen tls-endpoint, the raw channel binding data is the certificate of the server the client connected to, which will frequently be 500 bytes or more. If it is then the channel binding payload is the base64 encoded SHA-1 hash of the server certificate.

3.2. Initial Client Response

The SASL client response is formatted as an HTTP [RFC2616] request. The HTTP request is limited in that the path MUST be "/". In the OAUTH mechanism no query string is allowed. The following header lines are defined in the client response:

*User (OPTIONAL): Contains the user identifier being authenticated, and is provided to allow correct discovery information to be returned.

Host (REQUIRED): Contains the host name to which the client connected.

Authorization (REQUIRED):
An HTTP Authorization header.

The user name is provided by the client to allow the discovery information to be customized for the user, a given server could allow multiple authenticators and it needs to return the correct one. For instance, a large ISP could provide mail service for several domains who manage their own user information. For instance, users at foo-example.com could be authenticated by an OAuth service at https://oauth.foo-example.com/, and users at bar-example.com could be authenticated by https://oauth.bar-example.com, but both could be served by a hypothetical IMAP server running at a third domain, imap.example.net.

3.2.1. Query String in OAUTH-PLUS

In the OAUTH-PLUS mechanism the channel binding information is carried in the query string. OAUTH-PLUS defines following query parameter(s):

* **cbdata (REQUIRED):** Contains the base64 encoded channel binding data, properly escaped as an HTML query parameter value.

3.3. Server's Response

The server validates the response per the specification for the authorization scheme used. If the authorization scheme used includes signing of the request parameters the client must provide a complete HTTP style request that satisfies the data requirements for the scheme in use.
In the OAUTH-PLUS mechanism the server examines the channel binding data, extracts the channel binding unique prefix, and extracts the raw channel biding data based on the channel binding type used. It then computes it's own copy of the channel binding payload and compares that to the payload sent by the client in the query parameters of the tunneled HTTP request. Those two must be equal for channel binding to succeed.
The server responds to a successfully verified client message by completing the SASL negotiation. The authentication scheme MUST carry the user ID to be used as the authorization identity (identity to act as). The server MUST use that ID as the user being authorized, that is the user assertion we accept and not other information such as from the URL or "User:" header.
The server responds to failed authentication by sending discovery information in an HTTP style response with the HTTP status code set to 401, and then failing the authentication.
If channel binding is in use and the channel binding fails the server responds with a minimal HTTP response without discovery information and the HTTP 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.

3.4. Mapping to SASL Identities

Some OAuth mechanisms can provide both an authorization identity and an authentication identity. An example of this is OAuth 1.0a [RFC5849] where the consumer key (oauth_consumer_key) identifies the entity using to token which equates to the SASL authentication identity, and is authenticated using the shared secret. The authorization identity in the OAuth 1.0a case is carried in the token (per the requirement above), which SHOULD validated independently. The server MAY use a consumer key or other comparable identity in the OAuth authorization scheme as the SASL authentication identity. If an appropriate authentication identity is not available the server MUST use the identity asserted in the token.

3.5. Discovery Information

The server MUST send discovery information in response to a failed authentication exchange or a request with an empty Authorization header. If discovery information is returned it MUST include an authentication endpoint appropriate for the user. If the "User" header is present the discovery information MUST be for that user. Discovery information is provided by the server to the client to allow a client to discover the appropriate OAuth authentication and token endpoints. The client then uses that information to obtain the access token needed for OAuth authentication. The client SHOULD cache and re-use the user specific discovery information for service endpoints.
Discovery information makes use of both the WWW-Authenticate header as defined in HTTP Authentication: Basic and Digest Access Authentication [RFC2617] and Link headers as defined in [RFC5988]. The following elements are defined for discovery information:

WWW-Authenticate A WWW-Authenticate header for each authentication scheme supported by the server. Authentication scheme names are case insensitive. The following [RFC2617] authentication parameters are defined:

realm REQUIRED -- (as defined by RFC2617)

scope OPTIONAL -- A quoted string. This provides the client an OAuth 2 scope known to be valid for the resource.

oauth2-authenticator An [RFC5988] Link header specifying the [I-D.ietf-oauth-v2] authentication endpoint. This link has an OPTIONAL link-extension "scheme", if included this link applies ONLY to the specified scheme.

oauth2-token An [RFC5988] Link header specifying the [I-D.ietf-oauth-v2] token endpoint. This link has an OPTIONAL link-extension "scheme", if included this link applies ONLY to the specified scheme.

oauth-initiate (Optional) An [RFC5988] Link header specifying the OAuth1.0a [RFC5849] initiation endpoint. The server MUST send this if "OAuth" is included in the supported list of HTTP authentication schemes for the server.

oauth-authorize (Optional) An [RFC5988] Link header specifying the OAuth1.0a [RFC5849] authentication endpoint. The server MUST send this if "OAuth" is included in the supported list of HTTP authentication schemes for the server.

oauth-token (Optional) An [RFC5988] Link header specifying the OAuth1.0a [RFC5849] token endpoint. The server MUST send this if "OAuth" is included in the supported list of HTTP authentication schemes for the server. This link type has one link-extension "grant-types" which is a space separated list of the OAuth 2.0 grant types that can be used at the token endpoint to obtain a token.

Usage of the URLs provided in the discovery information is defined in the relevant specifications. If the server supports multiple authenticators the discovery information returned for unknown users MUST be consistent with the discovery information for known users to prevent user enumeration. The OAuth 2.0 specification [I-D.ietf-oauth-v2] supports multiple types of authentication schemes and the server MUST specify at least one supported authentication scheme in the discovery information. The server MAY support multiple schemes and MAY support schemes not listed in the discovery information.
If the resource server provides a scope the client SHOULD always request scoped tokens from the token endpoint. The client MAY use a scope other than the one provided by the resource server. Scopes other than those advertised by the resource server must be defined by the resource owner and provided in service documentation (which is beyond the scope of this memo).

3.6. Use of Signature Type Authorization

This mechanism supports authorization using signatures, which requires that both client and server construct the string to be signed. OAuth 2 is designed for authentication/authorization to access specific URIs. SASL is designed for user authentication, and has no facility for being more specific. In this mechanism we require an HTTP style format specifically to support signature type authentication, but this is extremely limited. The HTTP style request is limited to a path of "/". This mechanism is in the SASL model, but is designed 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.

```
GET / HTTP/1.1
Host: server.example.com
User: user@example.com
Authorization: MAC token="h480djs93hd8",timestamp="137131200",
               nonce="dj83hs9s",signature="YTVjyNSujYs1WsDurFnvFi4JK6o="
```

Using the example in the MAC specification [I-D.ietf-oauth-v2-http-mac] as a starting point, on an IMAP server running on port 143 and given the MAC style authorization request (with long lines wrapped for readability) below:

```
h480djs93hi8\n
137131200\n
dj83hs9s\n
\n
GET\n
server.example.com\n
143\n
/\n
\n
```

The normalized request string would be constructed per the MAC specification [I-D.ietf-oauth-v2-http-mac]. In this example the normalized request string with the new line separator character is represented by "\n" for display purposes only would be:

4. GSS-API OAuth Mechanism Specification

Note: The normative references in this section are informational for SASL implementers, but they are normative for GSS-API implementers. The SASL OAuth mechanism is also a GSS-API mechanism and the messages described in Section 3 are the same, but

1. the GS2 header on the client's first message is excluded when OAUTH is used as a GSS-API mechanism, and

2. initial context token header is prefixed to the client's first authentication message (context token), as described in Section 3.1 of RFC 2743,

The GSS-API mechanism OID for OAuth is [[TBD: IANA]].
OAuth security contexts always have the mutual_state flag (GSS_C_MUTUAL_FLAG) set to TRUE. OAuth supports credential delegation,

therefore security contexts may have the deleg_state flag (GSS_C_DELEG_FLAG) set to either TRUE or FALSE.

The mutual authentication property of this mechanism relies on successfully comparing the TLS server identity with the negotiated target name. Since the TLS channel is managed by the application outside of the GSS-API mechanism, the mechanism itself is unable to confirm the name while the application is able to perform this comparison for the mechanism. For this reason, applications MUST match the TLS server identity with the target name, as discussed in [RFC6125].

The OAuth mechanism does not support per-message tokens or GSS_Pseudo_random.

OAuth supports a standard generic name syntax for acceptors, such as GSS_C_NT_HOSTBASED_SERVICE (see [RFC2743], Section 4.1). These service names MUST be associated with the "entityID" claimed by the RP. OAuth supports only a single name type for initiators: GSS_C_NT_USER_NAME. GSS_C_NT_USER_NAME is the default name type. The query, display, and exported name syntaxes for OAuth principal names are all the same. There is no OAuth-specific name syntax; applications SHOULD use generic GSS-API name types, such as GSS_C_NT_USER_NAME and GSS_C_NT_HOSTBASED_SERVICE (see [RFC2743], Section 4). The exported name token does, of course, conform to [RFC2743], Section 3.2, but the "NAME" part of the token should be treated as a potential input string to the OAuth name normalization rules.

5. Examples

These example illustrate exchanges between an IMAP client and an IMAP server.

5.1. Successful Bearer Token Exchange

This example shows a successful OAuth 2.0 bearer token exchange with an initial client response. Note that line breaks are inserted for readability.

```
S: * IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTH
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH R0VUIC8gSFRUUC8xLjENCkhvc3Q6IGltYXAuZXhhbXBs
      ZS5jb20NCkF1dGhvcml6YXRpb246IEJFQVJFUiAidkY5ZGZ0NHFtVGMyTnZiM1J
      sY2tCaGJIUmhkbWx6ZEdFdVkyOXRDZz09Ig0KDQo=
S: +
S: t1 OK SASL authentication succeeded

GET / HTTP/1.1
Host: imap.example.com
Authorization: BEARER "vF9dft4qmTc2Nvb3RlckBhbHRhdmlzdGEuY29tCg=="
```

As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response is:
The line containing just a "+" and a space is an empty response from the server. This response contains discovery information, and in the success case no discovery information is necessary so the response is empty. Like other messages, and in accordance with the IMAP SASL binding, the empty response is base64-encoded.

5.2. MAC Authentication with Channel Binding

This example shows a channel binding failure. The example sends the same request as above, but in the context of an OAUTH-PLUS exchange the channel binding information is missing. Note that line breaks are inserted for readability.

```
S: * CAPABILITY IMAP4rev1 AUTH=OAUTH SASL-IR IMAP4rev1 Server Ready
S: t0 OK Completed
C: t1 AUTHENTICATE MAC R0VUIC8/Y2JkYXRhPSJTRzkzSUdKcFp5QnBjeUJoSUZSTVV5Q
   m1hVzVoYkNCdFpYTnpZV2RsUHdvPSIgSFRUUC8xLjENCkhvc3Q6IHNlcnZlci5leGFtcG
   xlLmNvbQ0KVXNlcjogdXNlckBleGFtcGxlLmNvbQ0KQXV0aG9yaXphdGlvbjogTUFDIHR
   va2VuPSJoNDgwZGpzOTNoZDgiLHRpbWVzdGFtcD0iMTM3MTMxMjAwIixub25jZT0iZGo4
   M2hzOXMiLHNpZ25hdHVyZT0iV1c5MUlHMTFjM1FnWW1VZ1ltOXlaV1F1SUFvPSINCg0K
S: +
S: t1 OK SASL authentication succeeded

GET /?cbdata="SG93IGJpZyBpcyBhIFRMUyBmaW5hbCBtZXNzYWdlPwo=" HTTP/1.1
Host: server.example.com
User: user@example.com
Authorization: MAC token="h480djs93hd8",timestamp="137131200",
               nonce="dj83hs9s",signature="WW91IG11c3QgYmUgYm9yZWQuIAo="
```

As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response is:
The line containing just a "+" and a space is an empty response from the server. This response contains discovery information, and in the success case no discovery information is necessary so the response is empty. Like other messages, and in accordance with the IMAP SASL binding, the empty response is base64-encoded.

5.3. Failed Exchange

This example shows a failed exchange because of the empty Authorization header, which is how a client can query for discovery information. Note that line breaks are inserted for readability.

```
S: * CAPABILITY IMAP4rev1 AUTH=OAUTH SASL-IR IMAP4rev1 Server Ready
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH R0VUIC8gSFRUUC8xLjENClVzZXI6IHNjb290ZXJAYW
   x0YXZpc3RhLmNvbQ0KSG9zdDogaW1hcC55YWhvby5jb20NCkF1dGhlbnRpY2F0ZT
   ogDQoNCg==
S: + SFRUUC8xLjEgNDAxIFVuYXV0aG9yaXplZA0KV1dXLUF1dGhlbnRpY2F0ZTogQk
   VBUkVSIHJlYWxtPSJleGFtcGxlLmNvbSINCkxpbms6IDxodHRwczovL2xvZ2luLn
   lhaG9vLmNvbS9vYXV0aD4gcmVsPSJvYXV0aDItYXV0aGVudGljYXRvciIgIA0KTG
   luazogPGh0dHBzOi8vbG9naW4ueWFob28uY29tL29hdXRoPiByZWw9Im91YXRoMi
   10b2tlbiINCg0K
S: t1 NO SASL authentication failed
```

```
GET / HTTP/1.1
User: alice@example.com
Host: imap.example.com
Authorization:
```

The decoded initial client response is:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: BEARER realm="example.com"
Link: <https://login.example.com/oauth> rel="oauth2-authenticator"
Link: <https://login.example.com/oauth> rel="oauth2-token"
```

The decoded server discovery response is:

5.4. Failed Channel Binding

This example shows a channel binding failure in a discovery request. The channel binding information is empty. Note that line breaks are inserted for readability.

```
S: * CAPABILITY IMAP4rev1 AUTH=OAUTH SASL-IR IMAP4rev1 Server Ready
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH R0VUIC8/Y2JkYXRhPSIiIEhUVFAvMS4xDQpVc2VyOi
   BhbGljZUBleGFtcGxlLmNvbQ0KSG9zdDogaW1hcC5leGFtcGxlLmNvbQ0KQXV0aG
   9yaXphdGlvbjoNCg0K
S: + SFRUUC8xLjEgNDEyIFByZWNvbmRpdGlvbiBGYWlsZWQNCg0KDQo=
S: t1 NO SASL authentication failed
```

```
GET /?cbdata="" HTTP/1.1
User: alice@example.com
Host: imap.example.com
Authorization:
```

The decoded initial client response is:

HTTP/1.1 412 Precondition Failed

The decoded server response is:

6. Security Considerations

This mechanism does not provide a security layer, but does provide a provision for channel binding. The OAuth 2 specification [I-D.ietf-oauth-v2] allows for a variety of usages, and the security properties of these profiles vary. The usage of bearer tokens, for example, provide security features similar to cookies. Applications using this mechanism SHOULD exercise the same level of care using this mechanism as they would in using the SASL PLAIN mechanism. In particular, TLS 1.2 or an equivalent secure channel MUST be implemented and its usage is RECOMMENDED.

Channel binding in this mechanism has different properties based on the authentication scheme used. Channel binding to TLS with a bearer token provides only a binding to the TLS layer. Authentication schemes like MAC tokens have a signature over the channel binding information. These provide additional protection against a man in the middle attacks, and the MAC authorization header is bound to the channel and only valid in that context.

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

An OAuth credential is not equivalent to the password or primary account credential. There are protocols like XMPP that allow actions like change password. The server SHOULD ensure that actions taken in the authenticated channel are appropriate to the strength of the presented credential.

It is possible for an application server running on Evil.example.com to tell a client to request a token from Good.example.org. A client following these instructions will pass a token from Good to Evil. This is by design, since it is possible that Good and Evil are merely names, not descriptive, and that this is an innocuous activity between cooperating two servers in different domains. For instance, a site might operate their authentication service in-house, but outsource their mail systems to an external entity.

Tokens have a lifetime associated with them. Reducing both the lifetime of a token provides security benefits in case that tokens leak. In addition a previously obtained token MAY 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 access credentials that appear to be valid.

7. IANA Considerations

7.1. SASL Registration

The IANA is requested to register the following SASL profile:

- *SASL mechanism profile: OAUTH
- *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: OAUTH-PLUS
- *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

7.2. GSS-API Registration

IANA is further requested to assign an OID for this GSS mechanism in the SMI numbers registry, with the prefix of iso.org.dod.internet.security.mechanisms (1.3.6.1.5.5) and to reference this specification in the registry.

7.3. Link Type Registration

Pursuant to [RFC5988] The following link type registrations [[will be]] registered by mail to link-relations@ietf.org.

7.3.1. OAuth 2 Authentication Endpoint

- *Relation Name: oauth2-authenticator
- *Description: An OAuth 2.0 authentication endpoint.
- *Reference:

*Notes: This link type indicates an OAuth 2.0 authentication endpoint that can be used for user authentication/authorization for the endpoint providing the link.

*Application Data: [optional]

7.3.2. OAuth 2 Token Endpoint

*Relation Name: oauth2-token

*Description: The OAuth token endpoint used to get tokens for access.

*Reference:

*Notes: The OAuth 2.0 token endpoint to be used for obtaining tokens to access the endpoint providing the link.

*Application Data: This link type has one link-extension "grant-types", which is the OAuth 2.0 grant types that can be used at the token endpoint to obtain a token. This is not an exclusive list, it provides a hint to the application of what SHOULD be valid. A token endpoint MAY support additional grant types not advertised by a resource endpoint.

7.3.3. OAuth 1.0a Request Initiation Endpoint

*Relation Name: oauth-initiate

*Description: The OAuth 1.0a request initiation endpoint used to get tokens for access.

*Reference:

*Notes: The OAuth 1.0a endpoint used to initiate the sequence, this temporary request is what the user approves to grant access to the resource.

*Application Data:

7.3.4. OAuth 1.0a Authorization Endpoint

*Relation Name: oauth-authorize

*Description: The OAuth 1.0a authorization endpoint used to approve an access request.

*Reference:

*Notes:

*Application Data:

7.3.5. OAuth 1.0a Token Endpoint

*Relation Name: oauth-token

*Description: The OAuth 1.0a token endpoint used to get tokens for access.

*Reference:

*Notes:

*Application Data:

8. Appendix A -- Document History

[[to be removed by RFC editor before publication as an RFC]]
-04

*Editorial clean-up and text in introduction improved.

*Added GSS-API support

-03

*Fixing channel binding, not tls-unique specific. Also defining how the CB data is properly generated.

*Various small editorial changes and embarassing spelling fixes.

-02

*Filling out Channel Binding

*Added text clarifying how to bind to the 2 kinds of SASL identities.

-01

*Bringing this into line with draft 12 of the core spec, the bearer token spec, and references the MAC token spec

*Changing discovery over to using the Link header construct from RFC5988.

*Added the seeds of channel binding.

-00

*Initial revision

9. References

9.1. Normative References

[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
[RFC2617]	Franks, J., Hallam-Baker, P.M., Hostetler, J.L., Lawrence, S.D., Leach, P.J., Luotonen, A. and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.
[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.
[RFC5056]	Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
[RFC5849]	Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.
[RFC5929]	Altman, J., Williams, N. and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.
[RFC5988]	Nottingham, M., "Web Linking", RFC 5988, October 2010.
[RFC2743]	Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
[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.
[RFC6125]	Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.
[I-D.ietf-oauth-v2]	Hammer-Lahav, E, Recordon, D and D Hardt, "The OAuth 2.0 Authorization Protocol", Internet-Draft draft-ietf-oauth-v2-22, September 2011.

[I-D.ietf-oauth-v2-bearer]	Jones, M, Hardt, D and D Recordon, "The OAuth 2.0 Authorization Protocol: Bearer Tokens", Internet-Draft draft-ietf-oauth-v2-bearer-14, November 2011.
[I-D.ietf-oauth-v2-http-mac]	Hammer-Lahav, E, Barth, A and B Adida, "HTTP Authentication: MAC Access Authentication", Internet-Draft draft-ietf-oauth-v2-http-mac-00, May 2011.

9.2. Informative References

[RFC3501]	Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.
[I-D.hammer-hostmeta]	Hammer-Lahav, E and B Cook, "Web Host Metadata", Internet-Draft draft-hammer-hostmeta-17, September 2011.
[I-D.jones-simple-web-discovery]	Jones, M and Y Goland, "Simple Web Discovery (SWD)", Internet-Draft draft-jones-simple-web-discovery-01, July 2011.

Authors' Addresses

William Mills Mills Yahoo! Inc. EMail: wmills@yahoo-inc.com

Tim Showalter Showalter Yahoo! Inc. EMail: timshow@yahoo-inc.com

Hannes Tschofenig Tschofenig Nokia Siemens Networks Linnoitustie 6 Espoo, 02600 Finland Phone: +358 (50) 4871445 EMail: Hannes.Tschofenig@gmx.net URI: http://www.tschofenig.priv.at