KITTEN
Internet-Draft
Intended status: Standards Track
Expires: December 1, 2012

W. Mills
Yahoo! Inc.
T. Showalter

H. Tschofenig
Nokia Siemens Networks

May 30, 2012

A SASL and GSS-API Mechanism for OAuth
draft-ietf-kitten-sasl-oauth-01

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. 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 December 1, 2012.

Copyright Notice

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].

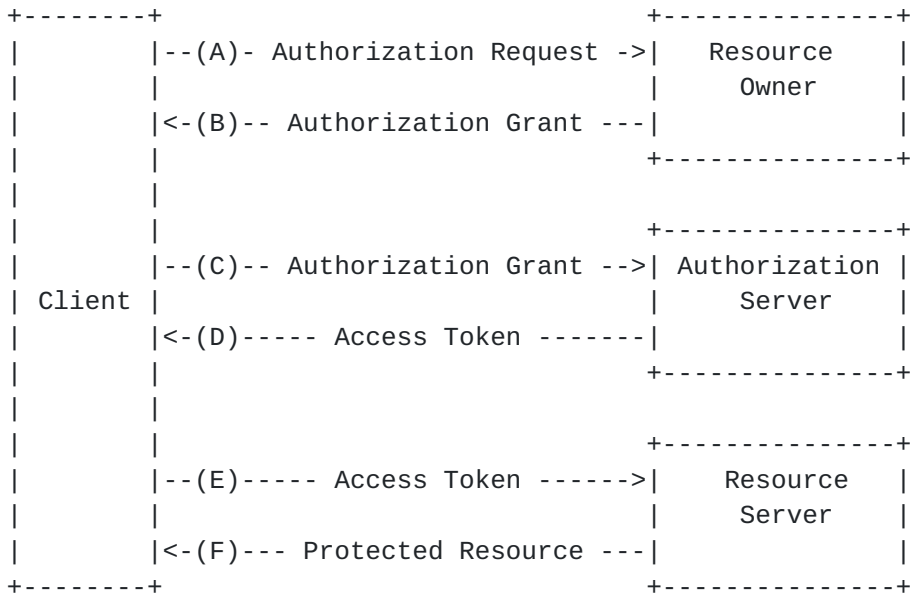

Figure 1: Abstract OAuth 2.0 Protocol Flow

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. Consequently, the message exchange shown in Figure 2 is the result of this specification. The client will genrally need to determine the authentication endpoints (and perhaps the service endpoints) before the OAuth 2.0 protocol exchange messages in steps (A)-(D) are executed. The discovery of the resource owner and authorization server endpoints is outside the scope of this specification. The client must discover those endpoints using a discovery mechanisms such as Webfinger using host-meta [I-D.jones-appsawg-webfinger]. 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.

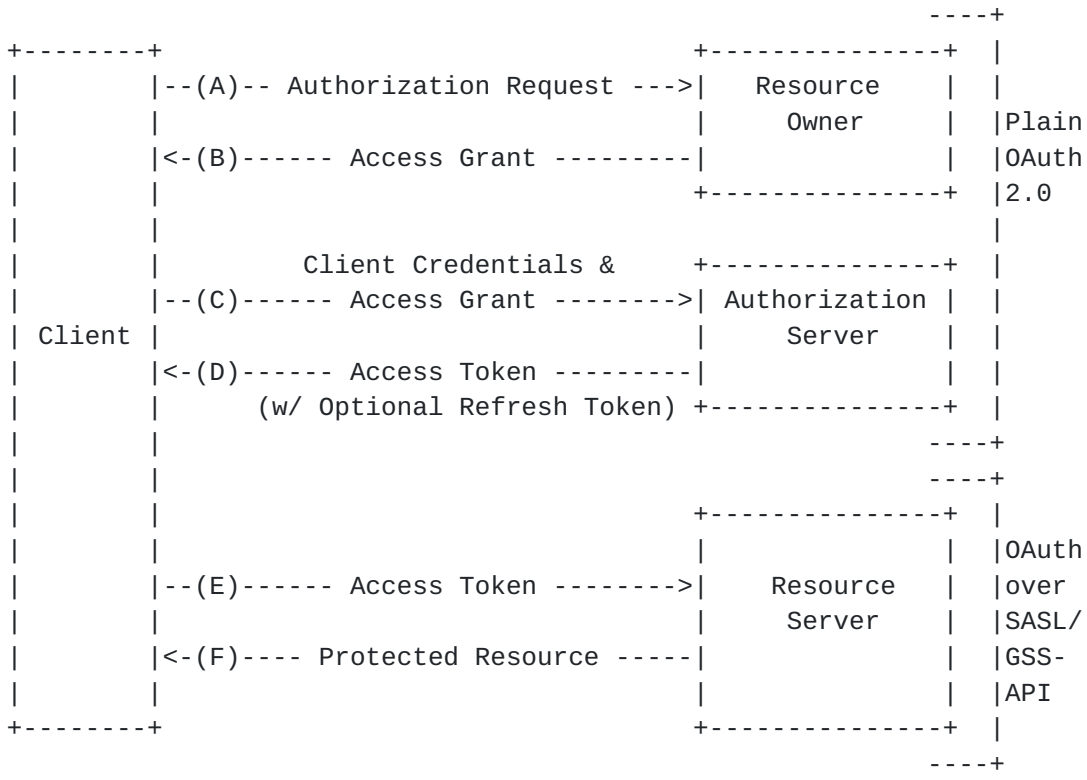

Figure 2: OAuth SASL Architecture

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 enables OAuth authorizattion schemes for SASL, "OAUTH-PLUS" adds channel binding [RFC5056] capability for additional security guarantees.

3.1. Initial Client Response

Client responses are a key/value pair sequence. These key/value pairs carry the equivalent values form an HTTP context in order to be able to complete an OAuth style HTTP authorization. The ABNF [RFC2234] syntax is:

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

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

auth (REQUIRED): The payload of the HTTP Authorization header for an equivalent HTTP OAuth authroization.

host: Contains the host name to which the client connected.

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

In authorization schemes that use signatures, the client MUST send host and port number key/values, and the server MUST fail authorization request requiring signatures that do not have host and port values.

3.1.1. Reserved Key/Values in OAUTH

In the OAUTH mechanism 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 schemes that use request signatures the default values MUST

be used unless explict values are provided in the client response. The following key values are reserved for future use:

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

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

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

3.2. Server's Response

The server validates the response per the specification for the authorization scheme used. If the authorization scheme used includes signing of the request parameters the client must provide a client response 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 cbdata key/value. 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 the ID obtained from the credential as the user being authorized.

3.2.1. 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, a value derived from it, 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 authorization identity as the wuthentication identity.

3.2.2. Server response to failed authentication.

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

status (REQUIRED): The authorization error code. Valid error codes are defined in the IANA [[need registry name]] registry specified in the OAuth 2 core specification.

scope (OPTIONAL): The OAuth scope required to access the service.

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 are be defined by the resource owner and provided in service documentation or discovery information (which is beyond the scope of this memo). If not present then the client SHOULD presume an empty scope (unscoped token) is needed.

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

3.3. 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 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 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 %x01 shown as ^A and long lines wrapped for readability) below:

```
host=server.example.com^A
port=143^A
auth=MAC token="h480djs93hd8",timestamp="137131200",nonce="dj83hs9s",
         signature="YTVjyNSujYs1WsDurFnvFi4JK6o="^A^A
```

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:

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

3.4. 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.

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 aG9zdD1zZXJ2ZXIuZXhhbXBsZS5jb20BcG9ydD0xNDMB
      YXV0aD1CRUFSRVIgdkY5ZGZ0NHFtVGMyTnZiM1JsY2tCaGJIUmhkbWx6ZEdFdVk
      yOXRDZz09AQE=
S: +
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:

```
host=server.example.com^Aport=143^A
auth=BEARER "vF9dft4qmTc2Nvb3RlckBhbHRhdmlzdGEuY29tCg=="^A^A
```

The line containing just a "+" and a space is an empty response from the server. This response contains error information, and in the success case the error 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 aG9zdD1zZXJ2ZXIuZXhhbXBsZS5jb20BcG9ydD0xNDMBYXV0a
   D1NQUMgdG9rZW49Img0ODBkanM5M2hkOCIsdGltZXN0YW1wPSIxMzcxMzEyMDAiLG5vbm
   NlPSJkajgzaHM5cyIsc2lnbmF0dXJlPSJZVFZqeU5TdWpZczFXc0R1ckZudkZpNEpLNm8
   9IgFjYmRhdGE9U0c5M0lHSnBaeUJwY3lCaElGUk1VeUJtYVc1aGJDQnRaWE56WVdkbFB3
   bz0BAQ==
S: +
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:

```
-
host=server.example.com^A
port=143^A
auth=MAC token="h480djs93hd8",timestamp="137131200",nonce="dj83hs9s",
         signature="YTVjyNSujYs1WsDurFnvFi4JK6o="^A
cbdata=SG93IGJpZyBpcyBhIFRMUyBmaW5hbCBtZXNzYWdlPwo=^A^A
```

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 the needed scope. 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 aG9zdD1zZXJ2ZXIuZXhhbXBsZS5jb20BcG9ydD0xND
   MBYXV0aD0BAQ==
S: + ewoic3RhdHVzIjoiNDAxIiwKInNjb3BlIjoiZXhhbXBsZV9zY29wZSIKfQo=
S: t1 NO SASL authentication failed
```

The decoded initial client response is:

```
host=server.example.com^Aport=143^Aauth=^A^A
```

The decoded server error response is:

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

5.4. Failed Channel Binding

This example shows a channel binding failure in an empty 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 aG9zdD1zZXJ2ZXIuZXhhbXBsZS5jb20BcG9ydD0xND
   MBYXV0aD0BY2JkYXRhPQEB
S: + ewoic3RhdHVzIjoiNDEyIiwKInNjb3BlIjoiZXhhbXBsZV9zY29wZSIKfQ==
S: t1 NO SASL authentication failed
```

The decoded initial client response is:

```
host=server.example.com^Aport=143^Aauth=^Acbdata=^A^A
```

The decoded server response is:

```
{
"status":"412",
"scope":"example_scope"
}
```

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 can implement 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.

Tokens have a lifetime associated with them. Reducing 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.

8. Appendix A -- Document History

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

-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.

-00

- Initial revision

9. References

9.1. Normative References

[I-D.ietf-oauth-v2]
Hammer-Lahav, E., Recordon, D., and D. Hardt, "The OAuth 2.0 Authorization Framework", draft-ietf-oauth-v2-26 (work in progress), May 2012.

[I-D.ietf-oauth-v2-bearer]
Jones, M., Hardt, D., and D. Recordon, "The OAuth 2.0 Authorization Protocol: Bearer Tokens", draft-ietf-oauth-v2-bearer-19 (work in progress), April 2012.

[I-D.ietf-oauth-v2-http-mac]
Hammer-Lahav, E., "HTTP Authentication: MAC Access Authentication", draft-ietf-oauth-v2-http-mac-01 (work in progress), February 2012.

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

[RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 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., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

[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.

[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.

[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.

[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.

[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.

9.2. Informative References

[I-D.jones-appsawg-webfinger]
Jones, P., Salgueiro, G., and J. Smarr, "WebFinger", draft-jones-appsawg-webfinger-05 (work in progress), May 2012.

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

Authors' Addresses

William Mills
Yahoo! Inc.

Phone:
Email: wmills@yahoo-inc.com

Tim Showalter

Phone:
Email: tjs@psaux.com

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at