KITTEN
Internet-Draft
Intended status: Standards Track
Expires: June 20, 2013

W. Mills
Yahoo! Inc.
T. Showalter

H. Tschofenig
Nokia Siemens Networks

December 17, 2012

A set of SASL and GSS-API Mechanisms for OAuth
draft-ietf-kitten-sasl-oauth-09

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) 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).

This Internet-Draft will expire on June 20, 2013.

Copyright Notice

Table of Contents

1. Introduction

OAuth [RFC6749] 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 2.0 specification [RFC6749] 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 separate specifications, for example bearer tokens [RFC6750], OAuth 2.0 MAC tokens [I-D.ietf-oauth-v2-http-mac]. OAuth 1.0a [RFC5849], the predecessor of OAuth 2.0, has a similar design. The main use cases for OAuth 2.0 and OAuth 1.0 have so far focused on an HTTP-based environment only.

Figure 1 shows the abstract message flow as shown in Figure 1 of OAuth 2.0 [RFC6749].

```
+--------+                               +---------------+
|        |--(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 ---|               |
+--------+                               +---------------+
```

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] and SMTP [RFC5321], 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 SASL mechanisms for OAuth, and it conforms to the new bridge between SASL and the GSS-API called GS2 [RFC5801]. This means that this document defines both SASL and GSS-API mechanisms. Implementers may be interested in either the SASL, the GSS-API, or even both mechanisms. To facilitate 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, indicates a successful authentication.

Steps (E) and (F) are not defined in [RFC6749] 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 generally need to determine the authentication endpoints (and perhaps the service endpoints) before the OAuth 2.0 protocol exchange messages in steps (A)-(D) are executed. The discovery of the resource owner and authorization server endpoints is outside the scope of this specification. The client must discover those endpoints using a discovery mechanisms such as Webfinger using host-meta [I-D.ietf-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.

```
                                                           ----+
+--------+                                +---------------+   |
|        |--(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) +---------------+   |
|        |                                                 ----+
|        |                                                 ----+
|        |                                +---------------+   |
|        |                                |               |   |OAuth
|        |--(E)------ Access Token -------->|    Resource   |   |over
|        |                                |     Server    |   |SASL/
|        |<-(F)---- Protected Resource -----|               |   |GSS-
|        |                                |               |   |API
+--------+                                +---------------+   |
                                                           ----+
```

Figure 2: OAuth SASL Architecture

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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

In examples, "C:" and "S:" indicate lines sent by the client and server respectively. Line breaks have been inserted for readability.

Note that the IMAP SASL specification requires base64 encoding, see Section 4 of [RFC4648], not this memo.

3. OAuth SASL Mechanism Specifications

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

OAUTHBEARER: Authorization using OAuth 2.0 bearer tokens as described in [RFC6750].

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

OAUTH10A-PLUS: Adds channel binding [RFC5056] capability to OAUTH10A for protection against man-in-the-middle attacks.

New extensions may be defined to add additional OAuth Access Token Types. Such a new SASL OAuth mechanism can be added by simply registering the new name(s) and citing this specification for the further definition. New channel binding enabled "-PLUS" mechanisms defined in this way MUST include message integrity protection. A newly defined mechanism would also need to register a new GS2 OID.

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

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

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

- Client sends an invalid initial client response.
- Server responds with an error message.
- Client sends a dummy client response.

o Server fails the authentication.

3.1. Initial Client Response

Client responses are a key/value pair sequence. The initial client response includes a gs2-header as defined in GS2 [RFC5801], which carries the authorization ID. These key/value pairs carry the equivalent values from an HTTP context in order to be able to complete an OAuth style HTTP authorization. Unknown key/value pairs MUST be ignored by the server. The ABNF [RFC5234] syntax is:

```
kvsep          = %x01
key            = 1*ALPHA
value          = *(VCHAR / SP / HTAB / CR / LF )
kvpair         = key "=" value kvsep
client_resp    = 0*kvpair kvsep
;; gs2-header  = As defined in GSS-API
initial_client_resp = gs2-header kvsep client_resp
```

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

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

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

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

qs: The HTTP query string. In non-channel binding mechanisms this is reserved, the client SHOUD NOT send it, and has the default value of "". In "-PLUS" variants this carries a single key value pair "cbdata" for the channel binding data payload formatted as an HTTP query string.

For OAuth Access Token Types that use digital signatures or keyed message digests the client MUST send host and port number key/values, and the server MUST fail an authorization request requiring signatures or keyed message digests that do not have host and port values. For authorization schemes that require a URI scheme as part of the data being signed "http" is always used. In OAuth 1.0a for example, the so-called signature base string calculation includes the reconstructed HTTP URL.

3.1.1. Reserved Key/Values

In these mechanisms values for path, query string and post body are assigned default values. OAuth authorization schemes MAY define usage of these in the SASL context and extend this specification. For OAuth Access Token Types that use request signatures 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 for use in signatures, the default value is "POST".

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

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

3.1.2. Use of the gs2-header

The OAuth scheme related mechanisms are also GSS-API mechanisms, see Section 4 for further detail. The gs2-header is used as follows:

- o The "gs2-nonstd-flag" MUST NOT be present.

- o The "gs2-authzid" carries the authorization identity as specified in [RFC5801]. If present the application MUST determine whether access is granted for the identity asserted in the OAuth credential, if it does not the server MUST fail the negotiation.

In the non "-PLUS" mechanisms the "gs2-cb-flag" MUST be set to "n" because channel-binding [RFC5056] data is not expected. In the OAUTH10A-PLUS mechanism (or other -PLUS variants based on this specification) the "gs2-cb-flag" MUST be set appropriately by the client.

3.2. Server's Response

The server validates the response per the specification for the OAuth Access Token Types used. If the OAuth Access Token Type utilizes a digital signature or 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.

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

3.2.1. OAuth Identifiers in the SASL Context

OAuth access tokens may carry the authenticated identifier of the resource owner and client authentication provides the authenticated identity of the client issuing the request to the resource server.

If both identities are needed by an application the developer will need to provide a way to communicate that from the SASL mechanism back to the application such as a GSS-API [RFC2743] named type like GSS_C_NT_USER_NAME or a comparable newly defined GSS-API name type or name attribute [RFC6680].

3.2.2. Server Response to Failed Authentication

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

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

scope (OPTIONAL): An OAuth scope which is valid to access the service. This may be empty which implies that unscoped tokens are required, or a space separated list. Use of a space separated list is NOT RECOMMENDED.

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

If channel binding is in use and the channel binding fails the server responds with a status code set to 412 to indicate that the channel

binding precondition failed. If the authentication scheme in use does not include signing the server SHOULD revoke the presented credential and the client SHOULD discard that credential.

3.2.3. Completing an Error Message Sequence

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

3.3. OAuth Access Token Types using Digital Signatures and Keyed Message Digests

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

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

Using the example in the OAuth 1.0a specification as a starting point, on an IMAP server running on port 143 and given the OAuth 1.0a style authorization request (with %x01 shown as ^A and line breaks added for readability) below:

```
n,a=user@example.com^A
host=example.com^A
user=user@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%3D"^A^A
```

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

- The method value is defaulted to POST.

- The scheme defaults to be "http", and any port number other than 80 is included.

- The path defaults to "/".

- The query string defaults to "".

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

```
POST&http%3A%2F%2Fexample.com:143%2F&oauth_consumer_key%3D9djdj82h4
8djs9d2%26oauth_nonce%3D7d8f3e4a%26oauth_signature_method%3DHMAC-SH
A1%26oauth_timestamp%3D137131201%26oauth_token%3Dkkk9d7dh3k39sjv7
```

3.4. Channel Binding

The channel binding data is carried in the "qs" (query string) key value pair formatted as a standard HTTP query parameter with the name "cbdata". Channel binding requires that the channel binding data be integrity protected end-to-end in order to protect against man-in-the-middle attacks. All SASL OAuth mechanisms with a "-PLUS" postfix MUST provide integrity protection. It should be noted that while the Bearer Access Token Type mandates TLS it does not create keying material at the application layer and is not suitable for use with channel bindings.

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. For example, if the client is using tls-unique for channel binding then the raw channel binding data is the TLS finished message as specified in Section 3.1 of [RFC5929].

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.

A SASL OAuth mechanism is also a GSS-API mechanism and the messages described in Section 3 are the same with the following changes to the GS2 related elements:

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

2. the 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 OIDs are:

o OAUTHBEARER: [[TBD: IANA -- probably in the 1.3.6.1.5.5 tree]]

o OAUTH10A: [[TBD: IANA -- probably in the 1.3.6.1.5.5 tree]]

OAuth mechanims 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 using the appropriate application profile, as discussed in [RFC6125]. For example, when SASL OAuth is run over IMAP then the IMAP profile of RFC 6125 is used.

OAuth mechanisms do 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 mechanisms support 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 examples illustrate exchanges between an IMAP and SMTP clients and servers.

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

5.1. Successful Bearer Token Exchange

This example shows a successful OAuth 2.0 bearer token exchange. 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 bixhPXVzZXJAZXhhbXBsZS5jb20BaG9zdD1zZX
      J2ZXIuZXhhbXBsZS5jb20BcG9ydD0xNDMBYXV0aD1CZWFyZXIgdkY5ZGZ0NHFtV
      GMyTnZiM1JsY2tCaGJIUmhkbWx6ZEdFdVkyOXRDZz09AQE=
S: t1 OK SASL authentication succeeded
```

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

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

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

```
[connection begins]
S: 220 mx.example.com ESMTP 12sm2095603fks.9
C: EHLO sender.example.com
S: 250-mx.example.com at your service,[172.31.135.47]
S: 250-SIZE 35651584
S: 250-8BITMIME
S: 250-AUTH LOGIN PLAIN OAUTHBEARER
S: 250-ENHANCEDSTATUSCODES
S: 250 PIPELINING
C: t1 AUTHENTICATE OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20BaG9zdD1zZX
      J2ZXIuZXhhbXBsZS5jb20BcG9ydD0xNDMBYXV0aD1CZWFyZXIgdkY5ZGZ0NHFtV
      GMyTnZiM1JsY2tCaGJIUmhkbWx6ZEdFdVkyOXRDZz09AQE=
S: 235 Authentication successful.
[connection continues...]
```

5.2. OAuth 1.0a Authorization with Channel Binding

This example shows channel binding in the context of an OAuth 1.0a request using a keyed message digest. Note that line breaks are inserted for readability.

```
S: * OK [CAPABILITY IMAP4rev1 AUTH=OAUTH10A-PLUS SASL-IR]
     IMAP4rev1 Server Ready
C: t1 AUTHENTICATE OAUTH10A-PLUS cD10bHMtdW5pcXVlLGE9dXNlckBleGFtcGxlL
     mNvbQFob3N0PXNlcnZlci5leGFtcGxlLmNvbQFwb3J0PTE0MwFhdXRoPU9BdXRoI
     HJlYWxtPSJFeGFtcGxlIixvYXV0aF9jb25zdW1lcl9rZXk9IjlkamRqODJoNDhka
     nM5ZDIiLG9hdXRoX3Rva2VuPSJra2s5ZDdkaDNrMzlzanY3IixvYXV0aF9zaWduY
     XR1cmVfbWV0aG9kPSJITUFDLVNIQTEiLG9hdXRoX3RpbWVzdGFtcD0iMTM3MTMxM
     jAxIixvYXV0aF9ub25jZT0iN2Q4ZjNlNGEiLG9hdXRoX3NpZ25hdHVyZT0iU1Nkd
     ElHRWdiR2wwZEd4bElIUmxZU0J3YjNRdSIBcXM9Y2JkYXRhPXRscy11bmlxdWU6U
     0c5M0lHSnBaeUJwY3lCaElGUk1VeUJtYVc1aGJDQnRaWE56WWdkbFB3bz0BAQ==
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:

```
p=tls-unique,a=user@example.com^A
host=server.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
qs=cbdata=tls-unique:SG93IGJpZyBpcyBhIFRMUyBmaW5hbCBtZXNzYWdlPwo=^A^A
```

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

```
POST&http%3A%2F%2Fserver.example.com:143%2F&cbdata=tls-unique:SG93I
GJpZyBpcyBhIFRMUyBmaW5hbCBtZXNzYWdlPwo=%26oauth_consumer_key%3D9djd
j82h48djs9d2%26oauth_nonce%3D7d8f3e4a%26oauth_signature_method%3DHM
AC-SHA1%26oauth_timestamp%3D137131201%26oauth_token%3Dkkk9d7dh3k39s
jv7
```

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=OAUTHBEARER SASL-IR IMAP4rev1 Server
     Ready
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTHBEARER cD10bHMtdW5pcXVlLGE9dXNlckBleGFtcG
      xlLmNvbQFob3N0PXNlcnZlci5leGFtcGxlLmNvbQFwb3J0PTE0MwFhdXRoP
      QFjYmRhdGE9AQE=
S: + ewoic3RhdHVzIjoiNDAxIgoic2NvcGUiOiJleGFtcGxlX3Njb3BlIgp9
C: + AQ==
S: t1 NO SASL authentication failed
```

The decoded initial client response is:

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

The decoded server error response is:

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

The client responds with the required dummy response.

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=OAUTH10A-PLUS SASL-IR IMAP4rev1 Server
     Ready
S: t0 OK Completed
C: t1 AUTHENTICATE OAUTH10A-PLUS cCxhPXVzZXJAZXhhbXBsZS5jb20BaG9z
     dD1zZXJ2ZXIuZXhhbXBsZS5jb20BcG9ydD0xNDMBYXV0aD0BY2JkYXRhPQEB
S: + ewoic3RhdHVzIjoiNDEyIiwKInNjb3BlIjoiZXhhbXBsZV9zY29wZSIKfQ==
C: + AQ==
S: t1 NO SASL authentication failed
```

The decoded initial client response is:

```
p=tls-unique,a=user@example.com,^Ahost=server.example.com^A
port=143^Aauth=^Acbdata=^A^A
```

The decoded server response is:

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

The client responds with the required dummy response.

5.5. SMTP Example of a Failed Negotiation

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

```
[connection begins]
S: 220 mx.example.com ESMTP 12sm2095603fks.9
C: EHLO sender.example.com
S: 250-mx.example.com at your service,[172.31.135.47]
S: 250-SIZE 35651584
S: 250-8BITMIME
S: 250-AUTH LOGIN PLAIN OAUTHBEARER
S: 250-ENHANCEDSTATUSCODES
S: 250 PIPELINING
C: AUTH OAUTHBEARER bixhPT1zb21ldXNlckBleGFtcGxlLmNvbQFhdXRoPUJlYXJlciB2
       RjlkZnQ0cW1UYzJOdmIzUmxja0JoZEhSaGRtbHpkR0V1WTI5dENnPT0BAQ==
S: 334 eyJzdGF0dXMiOiI0MDEiLCJzY2hlbWVzIjoiYmVhcmVyIG1hYyIsInNjb3BlIjoia
       HR0cHM6Ly9tYWlsLmdvb2dsZS5jb20vIn0K
C: AQ==
S: 535-5.7.1 Username and Password not accepted. Learn more at
S: 535 5.7.1 http://support.example.com/mail/oauth
[connection continues...]
```

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

6. Security Considerations

OAuth 1.0a and OAuth 2 allows for a variety of deployment scenarios, and the security properties of these profiles vary. Application developers therefore need to understand the needs of their applications before selecting a specific SASL OAuth mechanism.

The channel binding in this mechanism has different properties based on the Access Token Type used.

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

The OAuth access token (and related keying material) is not equivalent to the user's long term password. As such, care has to be taken when these OAuth credentials are used for actions like changing passwords (as it is possible with some protocols, e.g., XMPP). The server SHOULD ensure that actions taken in the authenticated channel are appropriate to the strength of the presented credential.

Access tokens have a lifetime. Reducing the lifetime of an access token provides security benefits, as described in [I-D.ietf-oauth-v2-threatmodel], and OAuth 2.0 introduces refresh tokens to obtain new access token on the fly. Additionally, a previously obtained access 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. Internationalization Considerations

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

At the time of writing the standardization of the assertion format (in JSON format) is still ongoing, see [I-D.ietf-oauth-json-web-token].

8. IANA Considerations

8.1. SASL Registration

The IANA is requested to register the following SASL profile:

SASL mechanism profile: OAUTHBEARER

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Owner/Change controller: the IETF

Note: None

The IANA is requested to register the following SASL profile:

SASL mechanism profile: OAUTH10A

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Owner/Change controller: the IETF

Note: None

The IANA is requested to register the following SASL profile:

SASL mechanism profile: OAUTH10A-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

8.2. GSS-API Registration

IANA is further requested to assign an OID for these GSS mechanisms 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.

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.

[RFC2473] Conta, A. and S. Deering, "Generic Packet Tunneling in IPv6 Specification", RFC 2473, December 1998.

[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.

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

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

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

[RFC6680] Williams, N., Johansson, L., Hartman, S., and S. Josefsson, "Generic Security Service Application Programming Interface (GSS-API) Naming Extensions", RFC 6680, August 2012.

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

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

9.2. Informative References

[I-D.ietf-appsawg-webfinger]
Jones, P., Salgueiro, G., and J. Smarr, "WebFinger", draft-ietf-appsawg-webfinger-07 (work in progress), December 2012.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-05 (work in progress), November 2012.

[I-D.ietf-oauth-v2-http-mac]
Richer, J., Mills, W., and H. Tschofenig, "OAuth 2.0 Message Authentication Code (MAC) Tokens", draft-ietf-oauth-v2-http-mac-02 (work in progress), November 2012.

[I-D.ietf-oauth-v2-threatmodel]
Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0

Threat Model and Security Considerations",
draft-ietf-oauth-v2-threatmodel-08 (work in progress),
October 2012.

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

Appendix A. Acknowlegements

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

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

Appendix B. Document History

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

-09

o Incorporated review by Alexey and Hannes.

o Clarified the three OAuth SASL mechanisms.

o Updated references

o Extended acknowledgements

-08

o Fixed the channel binding examples for p=$cbtype

o More tuning of the authcid language and edited and renamed 3.2.1.

-07

o Struck the MUST langiage from authzid.

o

-06

o Removed the user field. Fixed the examples again.

o Added canonicalization language.

o

-05

o Fixed the GS2 header language again.

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

o Added the user field back into the SASL message.

o Fixed the examples (again).

o

-04

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

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

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

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

- o Added a SMTP examples for Bearer use case.

-03

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

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

-02

- o Added the user data element back in.

- o Minor editorial changes.

-01

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

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

-00

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

- o Minor fixes.

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