

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: July 19, 2010

D. Hardt, Ed.
Microsoft
A. Tom
Yahoo!
B. Eaton
Google
Y. Goland
Microsoft
January 15, 2010

OAuth Web Resource Authorization Profiles
draft-hardt-oauth-01

Abstract

The OAuth Web Resource Authorization Profiles (OAuth WRAP) allow a server hosting a Protected Resource to delegate authorization to one or more authorities. An application (Client) accesses the Protected Resource by presenting a short lived, opaque, bearer token (Access Token) obtained from an authority (Authorization Server). There are Profiles for how a Client may obtain an Access Token when acting autonomously or on behalf of a User.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on July 19, 2010.

Copyright Notice

Copyright (c) 2010 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 BSD License.

Table of Contents

1. Overview	5
1.1. Accessing a Protected Resource	5
1.2. Autonomous Client Profiles	6
1.3. User Delegation Profiles	7
2. Requirements Language	9
3. Definitions	10
3.1. URLs	10
4. Accessing a Protected Resource	11
4.1. Access Token	11
4.2. Acquiring an Access Token	11
4.3. Client Calls Protected Resource Using HTTP Header	12
4.4. Client Calls Protected Resource Using URL Query Parameter	12
4.5. Client Calls Protected Resource Using Post Parameter	13
5. Acquiring an Access Token: Autonomous Client Profiles	13
5.1. Client Account and Password Profile	13
5.1.1. Provisioning	13
5.1.2. Client Requests Access Token	13
5.1.3. Successful Access Token Response from Authorization Server	14
5.1.4. Unsuccessful Access Token Response from Authorization Server	14
5.1.5. Client Refreshes Access Token	15
5.2. Assertion Profile	15
5.2.1. Provisioning	15
5.2.2. Client Obtains Assertion	15
5.2.3. Client Requests Access Token	15
5.2.4. Successful Access Token Response from Authorization Server	16
5.2.5. Unsuccessful Access Token Response from Authorization Server	16
5.2.6. Client Refreshes Access Token	16

6.	Acquiring an Access Token: User Delegation Profiles	17
6.1.	Username and Password Profile	17
6.1.1.	Provisioning	17
6.1.2.	Client Obtains Username and Password	17
6.1.3.	Client Requests Access Token	17
6.1.4.	Successful Access Token Response from Authorization Server	18
6.1.5.	Unsuccessful Access Token Response from Authorization Server	18
6.1.6.	Verification URL Response from Authorization Server	18
6.1.7.	CAPTCHA Response from Authorization Server	19
6.1.8.	Client Refreshes Access Token	19
6.1.9.	Successful Access Token Refresh	20
6.1.10.	Unsuccessful Access Token Refresh	20
6.2.	Web App Profile	20
6.2.1.	Provisioning	21
6.2.2.	Client Directs the User to the Authorization Server	21
6.2.3.	Authorization Server Confirms Authorization Request with User	21
6.2.4.	Authorization Server Directs User back to the Client	22
6.2.5.	Client Requests Access Token	22
6.2.6.	Successful Access Token Response from Authorization Server	23
6.2.7.	Unsuccessful Access Token Response from Authorization Server	24
6.2.8.	Client Refreshes Access Token	25
6.2.9.	Successful Access Token Refresh	25
6.2.10.	Unsuccessful Access Token Refresh	26
6.3.	Rich App Profile	26
6.3.1.	Provisioning	26
6.3.2.	Client Directs the User to the Authorization Server	26
6.3.3.	Authorization Server Confirms Authorization Request with User	27
6.3.4.	Client Requests Access Token	28
6.3.5.	Successful Access Token Response from Authorization Server	29
6.3.6.	Unsuccessful Access Token Response from Authorization Server	29
6.3.7.	Client Refreshes Access Token	30
6.3.8.	Successful Access Token Refresh	30
6.3.9.	Unsuccessful Access Token Refresh	30
7.	Parameter Considerations	31
7.1.	Authorization Server Request / Response Parameter Encoding	31
7.2.	Parameter Size	31

7.3.	Access Token Format	31
7.4.	Refresh Token Format	32
7.5.	Additional Authorization Server Parameters	32
7.6.	Parameter Names and Order	32
8.	IANA Considerations	32
9.	Security Considerations	32
10.	References	32
10.1.	Normative References	32
10.2.	Informative References	33
Appendix A.	Client Account and Password Profile Example	33
A.1.	Provisioning	33
A.2.	Client Requests Access Token	34
A.3.	Successful Access Token Response from Authorization Server	34
A.4.	Client Calls Protected Resource	35
Appendix B.	Web App Profile Example	35
B.1.	Provisioning	35
B.2.	Client Directs the User to the Server	36
B.3.	Authorization Server Confirms Delegation Request with User	36
B.4.	Server Directs User back to the Client	36
B.5.	Client Requests Access Token	37
B.6.	Successful Access Token Response from Authorization Server	37
B.7.	Client Calls Protected Resource	38
B.8.	Client Refreshes Access Token	38
Authors' Addresses	39

1. Overview

As the internet has evolved, there is a growing trend for a variety of applications (Clients) to access resources through an API over HTTP or other protocols. Often these resources require authorization for access and are Protected Resources. The systems that are trusted to make authorization decisions may be independent from the Protected Resources for scale and security reasons. The OAuth Web Resource Authorization Profiles (OAuth WRAP) enable a Protected Resource to delegate the authorization to access a Protected Resource to one or more trusted authorities.

Clients that wish to access a Protected Resource first obtain authorization from a trusted authority (Authorization Server). Different credentials and profiles can be used to obtain this authorization, but once authorized, the Client is provided an Access Token, and possibly a Refresh Token to obtain new Access Tokens. The Authorization Server typically includes authorization information in the Access Token and digitally signs the Access Token. Protected Resource can verify that an Access Token received from a Client was issued by a trusted Authorization Server and is valid. The Protected Resource can then examine the contents of the Access Token to determine the authorization that has been granted to the Client.

1.1. Accessing a Protected Resource

The Access Token is opaque to the Client, and can be any format agreed to between the Authorization Server and the Protected Resource enabling existing systems to reuse suitable tokens, or use a standard token format such as a Simple Web Token or JSON Web Token. Since the Access Token provides the Client authorization to the Protected Resource for the life of the Access Token, the Authorization Server should issue Access Tokens that expire within an appropriate time. When an Access Token expires, the Client requests a new Access Token from the Authorization Server, which once again computes the Client's authorization, and issues a new Access Token. Figure 1 below shows the flow between the Client and Authorization Server (A,B); and then between the Client and Protected Resource (C,D):

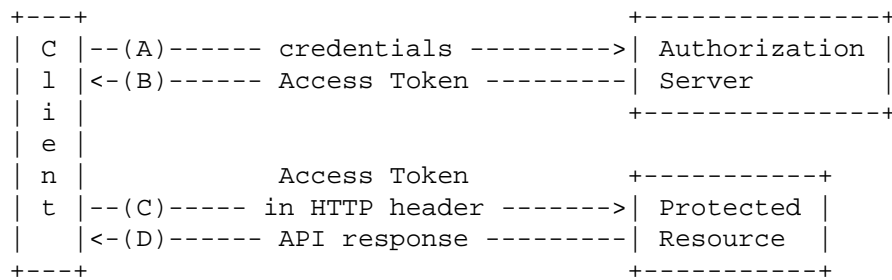


Figure 1

In step A, the Client presents credentials to the Authorization Server in exchange for an Access Token.

A Profile specifies the credentials to be provided in step A, and how the Client obtains them. This specification defines a number of Profiles; additional Profiles may be defined to support additional scenarios. The Profiles in this specification are separated into two groups: autonomous profiles where the Client is acting for itself, and user delegation profiles where the Client is acting on behalf of a User.

1.2. Autonomous Client Profiles

The following two Profiles (see [Section 5](#)) are recommended for scenarios involving a Client acting autonomously.

Client Account and Password Profile ([Section 5.1](#)): This is the simplest Profile. The Client is provisioned with an account name and corresponding password by the Authorization Server. The Client presents the account name and password to the Access Token URL at the Authorization Server in exchange for an Access Token. This Profile is not intended for a Client acting on behalf of a User. See the User Delegation Profiles.

Assertion Profile ([Section 5.2](#)): This profile enables a Client with a SAML [[OASIS.saml-core-2.0-os](#)] or other assertion recognized by the Authorization Server. The Client presents the assertion to the Access Token URL at the Authorization Server in exchange for an Access Token. How the Client obtains the assertion is out of scope of OAuth WRAP.

Access Tokens are short lived bearer tokens. When the Protected Resource is presented with an expired Access Token by the Client, the Protected Resource returns an error. The Client presents the assertion once again to the Authorization Server to obtain a new Access Token.

1.3. User Delegation Profiles

Common scenarios involve the User delegating to a Client to act on the User's behalf, adding another party (the User) to the protocol. In these Profiles (see [Section 6](#)), the Client receives a Refresh Token when it acquires the first Access Token. When an Access Token expires, the Client presents the Refresh Token to acquire a new Access Token. Refresh Tokens are sensitive as they represent long-lived permissions to access a Protected Resource and are always transmitted using HTTPS.

Username and Password Profile ([Section 6.1](#)): While the User may use a username and password to authenticate to the Authorization Server, it is undesirable for the Client to store the User's username and password. In this profile the User provides their username and password to an application (Client) they have installed on their device. The Client presents a Client Identifier, the username and password (credentials) to the Access Token URL at the Authorization Server in exchange for an Access Token and a Refresh Token as depicted in Figure 2 below.

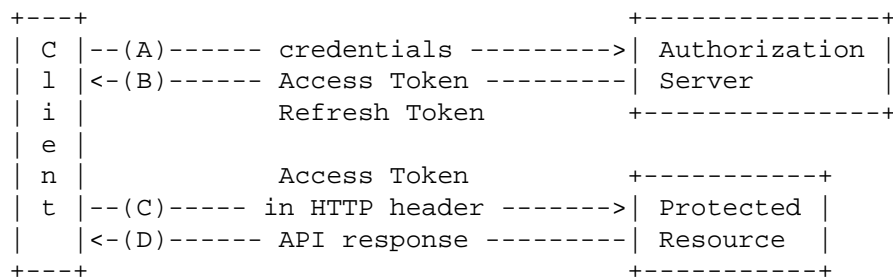


Figure 2

When the Access Token expires, the Client presents the Refresh Token to the Refresh Token URL at the Authorization Server in exchange for a new Access Token (Figure 3, steps A and B). The Client then uses the new Access Token to access the Protected Resource (Figure 3, steps C and D).

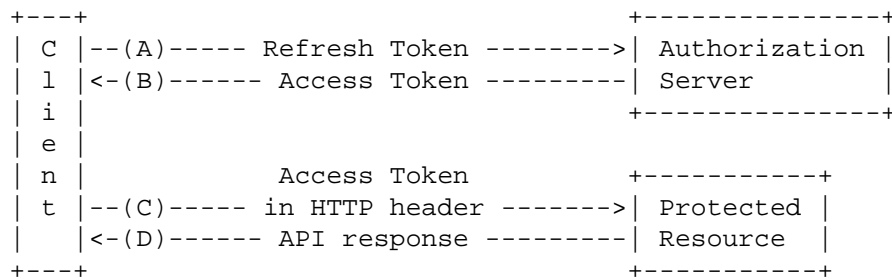


Figure 3

Web App Profile ([Section 6.2](#)): It is undesirable for a User to provide their Authorization Server username and password to web applications. Additionally, the User may authenticate to the Authorization Server using other mechanisms than a username and password. In this profile, a web application (Client) has been provisioned with a Client Identifier and Client Secret and may have registered a Callback URL. Figure 4 below illustrates the protocol. (A) The Client initiates the protocol by redirecting the User to the User Authorization URL at the Authorization Server passing the Client Identifier and the Callback URL. (B) The Authorization Server authenticates the User, confirms the User would like to authorize the Client to access the Protected Resource, and generates a Verification Code. (C) The Authorization Server then redirects the User to the Callback URL at the Client passing along the Verification Code.

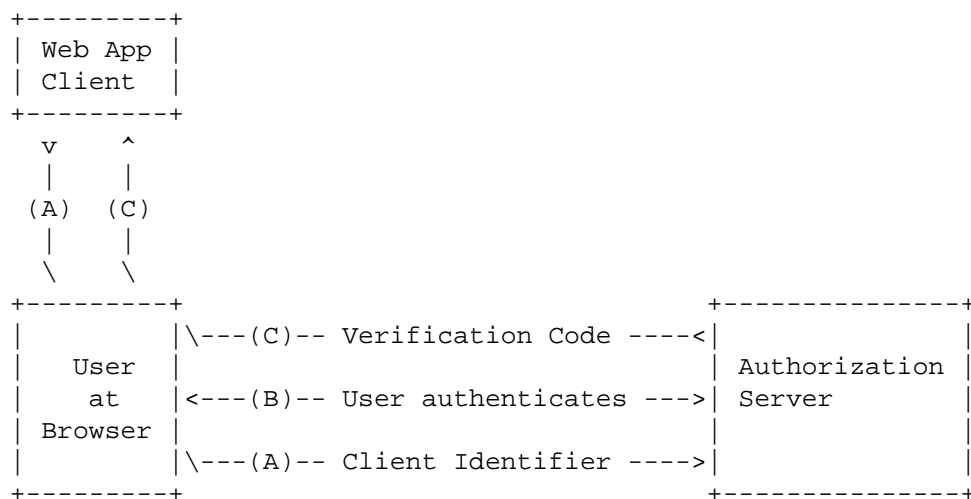


Figure 4

Similar to step A in Figure 2, the Client then presents the Client

Identifier, Client Secret, Callback URL and Verification code (credentials) to the Access Token URL at the Authorization Server for an Access Token and a Refresh Token.

Rich App Profile ([Section 6.3](#)): This profile is suitable when the Client is an application the User has installed on their device and a web browser is available, but it is undesirable for the User to provide their username and password to an application, or the user may not be using a username and password to authenticate to the Authorization Server.

The Client initiates the protocol by directing the User's browser to the Authorization URL at the Authorization Server passing the Client Identifier and potentially a Callback URL. The Authorization Server authenticates the User, confirms the User would like to authorize the Client to access the Protected Resource, and generates a Verification Code. The Verification Code may be communicated back to the Client in a number of ways:

- a. the Authorization Server presents the Verification Code to the User, who is instructed to enter the Verification Code directly in the Client;
- b. the Client reads the Verification Code from the title of the web page presented by the Authorization Server;
- c. the Authorization Server redirects the User to the Callback URL that presents Client specific language for the User to enter the Verification Code into the Client; or
- d. the Client has registered a custom scheme and the Authorization Server redirects the browser to the custom scheme that causes the User's browser to load the Client application with the Verification Code as a parameter.

Similar to step A in Figure 2, the Client then presents the Client Identifier, Callback URL (if provided) and Verification code (credentials) to the Access Token URL at the Authorization Server for an Access Token and a Refresh Token.

2. Requirements Language

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 [\[RFC2119\]](#). Domain name examples use [\[RFC2606\]](#).

3. Definitions

Access Token: a short lived bearer token issued by the Authorization Server to the Client. The Access Token is presented by the Client to the Protected Resource to access protected resources.

Authorization Server: an authorization resource that issues Access Tokens to Clients after successful authorization. May be the same entity as the Protected Resource.

Client: an application that would like access to a Protected Resource. **Client Identifier:** "> a value used by a Client to identify itself to the Authorization Server. This may be a human readable string or an opaque identifier.

Client Secret: a secret used by a web application Client to establish ownership of the Client Identifier.

Profile: a mechanism for a Client to obtain an Access Token from an Authorization Server.

Protected Resource: a protected API that allows access via OAuth WRAP. May be the same entity as the Authorization Server. **Refresh Token:** "> a long lived bearer token used by a Client to acquire an Access Token from an Authorization Server.

User: an individual who has an account with the Authorization Server.

Verification Code: a code used by a Client to verify the User has authorized the Client to have specific access to a Protected Resource.

3.1. URLs

Access Token URL: the Authorization Server URL at which an Access Token is requested by the Client. The URL may accept a variety of parameters depending on the Profile. A Refresh Token may also be returned to the Client. This URL MUST be an HTTPS URL and MUST always be called with POST.

Callback URL: the Client URL where the User will be redirected after an authorization request to the Authorization Server.

Refresh Token URL: the Authorization Server URL at which a Refresh Token is presented in exchange for a new Access Token is requested. This URL MUST be an HTTPS URL and MUST always be called with POST.

User Authorization URL: the Authorization Server URL where the Client redirects the User to make an authorization request.

4. Accessing a Protected Resource

Clients always present an Access Token to access a Protected Resource. Use of the Authorization header is RECOMMENDED, since HTTP implementations are aware that Authorization headers have special security properties and may require special treatment in caches and logs. Protected Resources SHOULD take precautions to insure that Access Tokens are not inadvertently logged or captured. In addition to the methods presented here, the Protected Resource MAY allow the Client to present the Access Token using any scheme agreed on by the Client and Protected Resource.

4.1. Access Token

The exact format of the Access Token is opaque to Clients and is out of scope of this specification. However, Protected Resources MUST be able to verify that the Access Token was issued by a trusted Authorization Server and is still valid. Access Tokens SHOULD periodically expire. The expiry time of Access Tokens is determined as an appropriate balance between excessive resource utilization if too short and unauthorized access if too long.

4.2. Acquiring an Access Token

An Authorization Server may support one or more protocol profiles that enable a Client to obtain an Access Token that can be used to access a Protected Resource.

Client developers only need to implement the profile(s) that align with how their application will be deployed and are supported by the Authorization Server.

Authorization Server developers only need to implement the profile(s) that are appropriate for them.

Protected Resource developers do not implement a profile as the Client always interacts with the Protected Resource by presenting an Access Token.

[Section 7](#) has general information about parameters passed to and from the Authorization Server.

See [Section 5](#) for how the Client acquires an Access Token when acting autonomously, and [Section 6](#) for how the Client acquires an Access

Token when acting acting on behalf of a User.

4.3. Client Calls Protected Resource Using HTTP Header

The Protected Resource SHOULD enable Clients to access the Protected Resource by including the Access Token in the HTTP Authorization header using the OAuth WRAP scheme with the following parameter:

access_token

REQUIRED. The value of the Access Token

For example, if the Access Token is the string 123456789, the HTTP header would be:

Authorization: WRAP access_token="123456789"

Note that per [section 1.2 of \[RFC2617\]](#) that the following header is also valid:

Authorization: WRAP access_token = 123456789

If the Access Token has expired or is invalid, the Protected Resource MUST return:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

4.4. Client Calls Protected Resource Using URL Query Parameter

The Protected Resource MAY allow the Client to access protected resources at the Protected Resource by including the following HTTP URL query parameter in the URL:

access_token

REQUIRED. The value of the Access Token

If the Access Token has expired or is invalid, the Protected Resource MUST return:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

4.5. Client Calls Protected Resource Using Post Parameter

The Protected Resource MAY allow the Client to access protected resources at the Protected Resource by including the following parameter in the body of a HTTP post message formatted as application/x-www-form-urlencoded per 17.13.4 of HTML 4.01 [W3C.REC-html40-19980424]:

access_token

REQUIRED. The value of the Access Token

If the Access Token has expired or is invalid, the Protected Resource MUST return:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

5. Acquiring an Access Token: Autonomous Client Profiles

These are the profiles the Client uses when acting autonomously.

5.1. Client Account and Password Profile

This profile is suitable when the Client is an application calling the Protected Resource on behalf of an organization and the Authorization Server accepts account passwords for authentication. This enables the Authorization Server to use an existing authentication mechanism. This profile SHOULD NOT be used when the Client is acting on behalf of a user. Profiles 6.1, 6.2 or 6.3 are RECOMMENDED when a Client is acting on behalf of a User.

5.1.1. Provisioning

Prior to initiating this protocol profile, the Client MUST have obtained an account name and account password from the Authorization Server.

5.1.2. Client Requests Access Token

The Client makes an HTTPS request to the Authorization Server's Access Token URL using POST. The request contains the following parameters:

wrap_name

REQUIRED. The account name.

wrap_password

REQUIRED. The account password.

wrap_scope

OPTIONAL. The Authorization Server MAY define authorization scope values for the Client to include.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

5.1.3. Successful Access Token Response from Authorization Server

If successful, the Authorization Server returns:

HTTP 200 OK

with the Refresh Token and an Access Token in the response body. The response body contains the following parameters:

wrap_refresh_token

REQUIRED. The Refresh Token.

wrap_access_token

REQUIRED. The Access Token.

wrap_access_token_expires_in

OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

The Client may now use the Access Token to access the Protected Resource per [Section 4](#)

5.1.4. Unsuccessful Access Token Response from Authorization Server

If the Client account name and password are invalid, the Authorization Server MUST respond with:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client MUST obtain a valid account name and password before retrying the request.

5.1.5. Client Refreshes Access Token

Authorization Servers SHOULD issue Access Tokens that expire and require Clients to refresh them. Upon receiving the HTTP 401 response when accessing protected resources per [Section 4](#), the Client should request a new Access Token by repeating [Section 5.1.2](#)

5.2. Assertion Profile

5.2.1. Provisioning

Prior to initiating this protocol profile, the Client MUST have a mechanism for obtaining an assertion from an assertion issuer that can be presented to the Authorization Server for access to the Protected Resource.

5.2.2. Client Obtains Assertion

The Client obtains an assertion. The process for obtaining the assertion is defined by the assertion issuer and the Authorization Server, and is out of scope of this specification.

5.2.3. Client Requests Access Token

The Client makes an HTTPS request to the Authorization Server's Access Token URL using POST. The request contains the following parameters:

wrap_assertion_format

REQUIRED. The format of the assertion as defined by the Authorization Server.

wrap_assertion

REQUIRED. The assertion.

wrap_scope

OPTIONAL. The Authorization Server MAY define authorization scope values for the Client to include

Additional parameters

Any additional parameters, as defined by the Authorization Server.

5.2.4. Successful Access Token Response from Authorization Server

If successful, the Authorization Server returns:

HTTP 200 OK

with the Access Token in the response body. The response body contains the following parameters:

wrap_access_token

REQUIRED. The Access Token.

wrap_access_token_expires_in

OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

The Client may now use the Access Token to access the Protected Resource per [Section 4](#).

5.2.5. Unsuccessful Access Token Response from Authorization Server

If the assertion is not valid, the Authorization Server MUST respond with:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client MUST obtain a valid assertion by repeating [Section 5.2.2](#) before retrying the request.

5.2.6. Client Refreshes Access Token

Authorization Servers SHOULD issue Access Tokens that expire and require Clients to refresh them. Upon receiving the HTTP 401 response when accessing protected resources per [Section 4](#), the Client should request a new Access Token by repeating [Section 5.2.3](#) if the assertion is still valid, otherwise the Client MUST obtain a new, valid assertion by repeating [Section 5.2.2](#).

6. Acquiring an Access Token: User Delegation Profiles

These are the profiles the Client uses when acting on behalf of a User.

6.1. Username and Password Profile

This profile is suitable where the Client is an application the User has installed on their computer and the User uses a username and password to authenticate to the Authorization Server. This profile enables a Client to act on behalf of the User without having to permanently store the User's username and password.

6.1.1. Provisioning

Prior to initiating this protocol profile, the Authorization Server MAY have required the Client to have obtained a Client Identifier from the Authorization Server.

6.1.2. Client Obtains Username and Password

The Client obtains the User's username and password from the user. The Client MUST discard the username and password once an Access Token has been obtained.

6.1.3. Client Requests Access Token

The Client makes an HTTPS request to the Authorization Server's Access Token URL using POST. The request contains the following parameters:

wrap_client_id
REQUIRED. The Client Identifier.

wrap_username
REQUIRED. The User's username.

wrap_password
REQUIRED. The User's password.

wrap_scope
OPTIONAL. The Authorization Server MAY define authorization scope values for the Client to include.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

6.1.4. Successful Access Token Response from Authorization Server

If successful, the Authorization Server returns:

HTTP 200 OK

with the Access Token in the response body. The response body contains the following parameters:

wrap_access_token

REQUIRED. The Access Token.

wrap_access_token_expires_in

OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

The Client MUST discard the User's username and password. The Client securely stores the Refresh Token for later use. The Client may now use the Access Token to access the Protected Resource per [Section 4](#).

6.1.5. Unsuccessful Access Token Response from Authorization Server

The Authorization Server MUST verify User's username and password. If the verification fails, the Authorization Server MUST respond with:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client needs to obtain a valid username and password from the User per [Section 6.1.2](#) before retrying the request.

6.1.6. Verification URL Response from Authorization Server

If the Authorization Server determines that the Client may be malicious, the Authorization Server MAY require the Client to instruct the User to visit a Verification URL. The Authorization Server communicates its requirement by responding to the Client's Access Token request with the following:

HTTP 400 Bad Request

and the body of the Authorization Server response contains the following parameter:

`wrap_verification_url`

REQUIRED. The verification URL that the Client MUST either load in the User's browser, or display for the User to enter into a browser.

The Client MUST then wait for the User to indicate they have successfully completed the verification process at the Authorization Server and attempt to obtain an Access Token Refresh Token per [Section 6.1.3](#) again.

6.1.7. CAPTCHA Response from Authorization Server

If the Authorization Server determines that the Client may be malicious, the Authorization Server MAY require the Client to have the User solve a CAPTCHA Puzzle. The Authorization Server communicates its requirement by responding to the Client's Access Token request with the following:

HTTP 400 Bad Request

and the body of the Authorization Server response contains the following parameter:

`wrap_captcha_url`

REQUIRED. The URL to the CAPTCHA puzzle image.

The Client MUST present the User with the CAPTCHA puzzle and prompt for a solution. The Client then MAY attempt to obtain an Access Token per [Section 6.1.3](#) again, including the following additional parameter:

`wrap_captcha_url`

REQUIRED. The URL to the CAPTCHA puzzle received from the Authorization Server.

`wrap_captcha_solution`

REQUIRED. The solution string to the CAPTCHA puzzle as defined by the Authorization Server.

6.1.8. Client Refreshes Access Token

Refreshing an Access Token is the same in [Section 6.1](#), [Section 6.2](#), and [Section 6.3](#). Authorization Servers SHOULD issue Access Tokens that expire and require Clients to refresh them. Upon receiving the HTTP 401 response when accessing protected resources per [Section 4](#),

the Client makes an HTTPS request to the Authorization Server's Refresh Token URL using POST. The request contains the following parameters:

`wrap_refresh_token`

REQUIRED. The Refresh Token that was received in [Section 6.1.3](#)

Additional parameters:

Any additional parameters, as defined by the Authorization Server.

6.1.9. Successful Access Token Refresh

If successful, the Authorization Server returns:

HTTP 200 OK

with the Access Token in the response body. The response body contains the following parameters:

`wrap_access_token`

REQUIRED. The Access Token.

`wrap_access_token_expires_in`

OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

6.1.10. Unsuccessful Access Token Refresh

The Authorization Server MUST verify the Refresh Token. If the verification fails, the Authorization Server MUST respond with

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client MUST again request authorization from the User by prompting for the User's username and password per [Section 6.1.2](#) before retrying the request.

6.2. Web App Profile

This profile is suitable when the Client is a web application calling the Protected Resource on behalf of a User. This profile enables a

Client to act on behalf of the User without acquiring a User's credentials.

6.2.1. Provisioning

Prior to initiating this protocol profile, the Client MUST have obtained a Client Identifier and Client Secret from the Authorization Server. The Authorization Server MAY have also required the Client to register the Callback URL.

6.2.2. Client Directs the User to the Authorization Server

The Client initiates an authorization request by redirecting the User's browser to the Authorization Server's User Authorization URL, with the following parameters:

wrap_client_id

REQUIRED. The Client Identifier.

wrap_callback

REQUIRED. The Callback URL. An absolute URL to which the Authorization Server will redirect the User back after the User has approved the authorization request. Authorization Servers MAY require that the wrap_callback URL match the previously registered value for the Client Identifier.

wrap_client_state

OPTIONAL. An opaque value that Clients can use to maintain state associated with this request. If this value is present, the Authorization Server MUST return it to the Client's Callback URL.

wrap_scope

OPTIONAL. The Authorization Server MAY define authorization scope values for the Client to include.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

6.2.3. Authorization Server Confirms Authorization Request with User

Upon receiving an authorization request from the Client by a redirection of the User's browser, the Authorization Server authenticates the user, presents the User with the Protected Resource access that will be granted to the Client, and prompts the User to confirm the request.

If the User denies the request, the Authorization Server MAY allow the User to return to the Client Callback URL with the following

parameters added:

wrap_error_reason

REQUIRED. Value is user_denied

wrap_client_state

REQUIRED if the Client sent the value in the authorization request in [Section 6.2.2](#)

If the User approves the request, the Authorization Server generates a Verification Code and associates it with the Client Identifier and Callback URL.

6.2.4. Authorization Server Directs User back to the Client

If the User approved the request, the Authorization Server MUST redirect the User back to the Callback URL, with the following parameters added:

wrap_verification_code

REQUIRED. The Verification Code.

wrap_client_state

REQUIRED if the Client sent the value in the authorization request in [Section 6.2.2](#)

Additional parameters:

Any additional parameters, as defined by the Authorization Server.

6.2.5. Client Requests Access Token

The Client makes an HTTPS request to the Authorization Server's Access Token URL, using POST. The request contains the following parameters in the body of the request:

wrap_client_id

REQUIRED. The Client Identifier

wrap_client_secret

REQUIRED. The Client Secret

wrap_verification_code

REQUIRED. The Verification Code.

wrap_callback

REQUIRED. The Callback URL used to obtain the Verification Code.

Additional parameters:

Any additional parameters, as defined by the Authorization Server.

6.2.6. Successful Access Token Response from Authorization Server

After receiving the Access Token request, the Authorization Server verifies the request as follows:

the Client Secret MUST match the Client Identifier

the Client Identifier MUST match the Client Identifier from the authorization redirect

the Verification Code MUST match the Client Identifier from the authorization redirect

the Callback URL MUST match the Callback URL from the authorization redirect

if the Callback URL or Callback URL pattern was registered with the Authorization Server, the Callback URL MUST match the Callback URL or Callback URL pattern as defined by the Authorization Server

the Verification Code MUST not have expired

The Authorization Server MAY also require that a Verification Code is not reused.

If verification is successful, the Authorization Server returns:

HTTP 200 OK

with the Refresh Token and the Access Token in the response body. The response body contains the following parameters:

wrap_refresh_token

REQUIRED. The Refresh Token.

wrap_access_token

REQUIRED. The Access Token.

wrap_access_token_expires_in

OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

The Client securely stores the Refresh Token for later use. The Client may now use the Access Token to access the Protected Resource per [Section 4](#).

6.2.7. Unsuccessful Access Token Response from Authorization Server

The Authorization Server MUST first verify the Client Identifier and Client Secret. If they are invalid, the Authorization Server MUST respond with:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client MUST obtain a valid Client Identifier and Client Secret before retrying the request.

The Authorization Server MUST then verify that the Callback URL and Verification Code are associated with the Client Identifier. If the verification fails, the Authorization Server MUST respond with:

HTTP 400 Bad Request

and the body of the Authorization Server response contains the following parameters:

wrap_error_reason

OPTIONAL. If all the parameters are valid except that the Verification Code has expired or been revoked, then it is RECOMMENDED that this parameter be included and if so, then the value MUST be:

expired_verification_code

This enables the Client to detect it needs a new Verification Code and to direct the User to the Authorization Server per [Section 6.2.2](#)

If the Callback URL is invalid, the value MUST be:

invalid_callback

Additional parameters

Any additional parameters, as defined by the Authorization Server.

6.2.8. Client Refreshes Access Token

Refreshing an Access Token is the same in [Section 6.1](#), [Section 6.2](#), and [Section 6.3](#). Authorization Servers SHOULD issue Access Tokens that expire and require Clients to refresh them. Upon receiving the HTTP 401 response when accessing protected resources per [Section 4](#), the Client makes an HTTPS request to the Authorization Server's Refresh Token URL using POST. The request contains the following parameters:

wrap_refresh_token

REQUIRED. The Refresh Token that was received in [Section 6.2.5](#)

Additional parameters:

Any additional parameters, as defined by the Authorization Server.

6.2.9. Successful Access Token Refresh

If successful, the Authorization Server returns:

HTTP 200 OK

with the Access Token in the response body. The response body contains the following parameters:

wrap_access_token

REQUIRED. The Access Token.

wrap_access_token_expires_in

OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

6.2.10. Unsuccessful Access Token Refresh

The Authorization Server MUST verify the Refresh Token. If the verification fails, the Authorization Server MUST respond with

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client MUST again request authorization from the User per [Section 6.2.2](#).

6.3. Rich App Profile

This profile is suitable where the Client is an application the User has installed on their computer and there is a browser available for the Client to launch. This profile enables a Client to act on behalf of the User regardless of how the User authenticates to the Server and without access to the User's credentials.

6.3.1. Provisioning

Prior to initiating this protocol profile, the Client MAY be required to register the Client Identifier and/or the Callback URL with the Server.

6.3.2. Client Directs the User to the Authorization Server

The Client initiates an authorization request by opening the User's browser with the Server's User Authorization URL, and including the following parameters:

wrap_client_id

REQUIRED. The Client Identifier.

wrap_callback

OPTIONAL. A Callback URL where the Authorization Server MAY redirect the User's browser after the User responds to the authorization request.

wrap_client_state

OPTIONAL. An opaque value that Clients can use to maintain state associated with this request. If this value is present, the Authorization Server MUST return it to the Client's Callback URL.

wrap_scope

OPTIONAL. The Authorization Server MAY define authorization scope values for the Client to include.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

6.3.3. Authorization Server Confirms Authorization Request with User

Upon receiving an authorization request from the Client by way of the User's browser, the Authorization Server authenticates the user, presents the User with the Protected Resource access that will be granted to the Client, and prompts the User to confirm the request. If the User approves the request, the Authorization Server generates a Verification Code. If the User denied access, the Authorization Server MAY set the Verification Code to the reserved value:

user_denied

It is RECOMMENDED the Verification Code be single use, and expire within minutes of issue. There are a number of mechanisms for the Authorization Server to transmit the Verification Code to the Client, specified below.

Rich Application interaction with the User and the Authorization Server is an area of active research and development. If the Rich Application is able to retrieve the verifier directly from the callback URL returned by the Authorization Server, an improved user experience is possible. However, not all applications are able to interact with the Authorization Server in this manner.

6.3.3.1. Applications with Callback URLs

Rich Applications may be able to receive callback URLs in any of several ways. For example, the Rich Application may register a custom protocol handler with the application platform so that the application will be invoked when the browser is redirected to the callback URL. Alternatively, the callback URL may point to a web site with which the Rich Application has a trust relationship. The web site can then pass the Callback URL down to the Rich Application for processing. Finally, the Callback URL may point to a web site that will display the Callback URL to the screen along with instructions for the user to enter the Verification Code into the application.

For Rich Applications with a Callback URL, the Authorization Server MUST redirect the User back to the Callback URL, with the following parameters added:

wrap_verification_code

REQUIRED. The Verification Code

wrap_client_state

REQUIRED if the Client sent the value in the authorization request in [Section 6.3.2](#)

Additional parameters

Any additional parameters, as defined by the Authorization Server.

If the User denied access, the Server MAY redirect the User's browser to the Callback URL with the Verification Code set to the reserved value:

user_denied

6.3.3.2. Applications without Callback URLs

Rich Applications without Callback URLs need to receive the verification code in other ways. For Rich Applications without a Callback URL, the Authorization Server MUST present the Verification Code on the web page and instruct the user to enter it into the Client.

The Server MAY also append the Verification Code to the title of the HTML page so that Clients that have access to the title of the browser's current page can obtain the Verification Code without requiring the User enter the Verification Code into the Client. The Client can parse the title looking for "code=" and then the rest of the title is the Verification Code. If adding the Verification Code to the title of the HTML page, the Server MUST also include the wrap_client_state parameter if sent from the Client as the "state=" parameter.

Eg. For example.com where the Verification Code = WF34F7HG and Client State = NMMGFJJ, the Server would set the title of the page to something like:

```
<title>Successful delegation, code=WF34F7HG  
state=NMMGFJJ</title>
```

If the User denied access, the Server MAY append code=user_denied to the title of the HTML page so that the Client can detect that the User has denied access.

6.3.4. Client Requests Access Token

The Client makes an HTTPS request to the Server's Access Token URL using POST. The request contains the following parameters:

wrap_client_id
REQUIRED. The Client Identifier

wrap_verification_code
REQUIRED. The Verification Code.

Additional parameters:

Any additional parameters, as defined by the Authorization Server.

6.3.5. Successful Access Token Response from Authorization Server

The Server checks the Verification Code was previously issued to the same Client Identifier, has not expired and has not been used. If these conditions are met, the Server marks the Verification Code as being used and returns:

HTTP 200 OK

with the Refresh Token and an Access Token in the response body. The response body contains the following parameters:

wrap_refresh_token
REQUIRED. The Refresh Token.

wrap_access_token
REQUIRED. The Access Token.

wrap_access_token_expires_in
OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

The Client securely stores the Refresh Token for later use. The Client may now use the Access Token to access the Protected Resource per [Section 4](#).

6.3.6. Unsuccessful Access Token Response from Authorization Server

The Authorization Server MUST first verify the Client Identifier and Client Secret per [Section 6.3.5](#). If they are invalid, the Authorization Server MUST respond with:

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client needs to obtain a new Verification Code per [Section 6.3.2](#).

6.3.7. Client Refreshes Access Token

Refreshing an Access Token is the same in [Section 6.1](#), [Section 6.2](#), and [Section 6.3](#). Authorization Servers SHOULD issue Access Tokens that expire and require Clients to refresh them. Upon receiving the HTTP 401 response when accessing protected resources per [Section 4](#), the Client makes an HTTPS request to the Authorization Server's Refresh Token URL using POST. The request contains the following parameters:

wrap_refresh_token

REQUIRED. The Refresh Token that was received in [Section 6.3.4](#)

Additional parameters:

Any additional parameters, as defined by the Authorization Server.

6.3.8. Successful Access Token Refresh

If successful, the Authorization Server returns:

HTTP 200 OK

with the Access Token in the response body. The response body contains the following parameters:

wrap_access_token

REQUIRED. The Access Token.

wrap_access_token_expires_in

OPTIONAL. The lifetime of the Access Token in seconds. For example, 3600 represents one hour.

Additional parameters

Any additional parameters, as defined by the Authorization Server.

6.3.9. Unsuccessful Access Token Refresh

The Authorization Server MUST verify the Refresh Token. If the verification fails, the Authorization Server MUST respond with

HTTP 401 Unauthorized

and the HTTP header:

WWW-Authenticate: WRAP

The Client MUST again request authorization from the User per [Section 6.3.2](#).

7. Parameter Considerations

7.1. Authorization Server Request / Response Parameter Encoding

All requests made directly to the Authorization Server use the HTTP POST method and the parameters MUST be in the body of the message and formatted as application/x-www-form-urlencoded per 17.13.4 of HTML 4.01 [[W3C.REC-html40-19980424](#)].

Any parameters in the response from the Authorization Server MUST be in the body of the message and formatted as application/x-www-form-urlencoded per 17.13.4 of HTML 4.01 [[W3C.REC-html40-19980424](#)].

7.2. Parameter Size

HTTP Headers

Web servers often impose a maximum on the combined size of all HTTP headers ranging from 8KB to 16KB. The size of the Access Token should be small enough to ensure the total size of the HTTP headers does not exceed the limits of web servers.

URLs

Web servers and browsers often impose a maximum on the total length of the URL of as low as 2083 bytes. The length of URLs exposed by the Authorization Server and the length of parameters passed on a URL should be minimized so that the total length does not exceed this limit.

7.3. Access Token Format

OAuth WRAP does not specify the format of the Access Token. The format is mutually agreed to by the Authorization Server and the Protected Resource and is opaque to the Client. The Access Token format MUST consist of legal characters in an HTTP header per [Reference needed]

The Simple Web Token (SWT) and JSON Web Token (JWT) are possible Access Token formats.

[TBD: entropy recommendations for Access Token so that it remains secure during its lifetime]

7.4. Refresh Token Format

OAuth WRAP does not specify the format of the Refresh Token. The Refresh Token is both generated and consumed by the Authorization Server and is opaque to the Client and never exposed to the Protected Resource. The Refresh Token is a long lived credential, and should contain enough entropy that it cannot be guessed. The size limitations of the Access Token are not applicable to the Refresh Token as the Refresh Token is always in the body of an HTTP message.

7.5. Additional Authorization Server Parameters

The Authorization Server may define additional parameters to be included in are returned from calls to the Access Token URL or User Authorization URL. Parameters that start with wrap_ are reserved and may not be used.

7.6. Parameter Names and Order

All parameter names are case sensitive. The parameters may appear in any order. Unrecognized parameters are allowed, but MUST be ignored.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

TBD: need to put in all the security considerations for implementors.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), March 1997.
- [RFC2606] Eastlake, D. and A. Panitz, "Reserved Top Level DNS Names", [BCP 32](#), [RFC 2606](#), 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.

[W3C.REC-html40-19980424]

Hors, A., Jacobs, I., and D. Raggett, "HTML 4.0 Specification", World Wide Web Consortium Recommendation REC-html40-19980424, April 1998, <<http://www.w3.org/TR/1998/REC-html40-19980424>>.

10.2. Informative References

[I-D.narten-iana-considerations-rfc2434bis]

Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [draft-narten-iana-considerations-rfc2434bis-09](#) (work in progress), March 2008.

[OASIS.saml-core-2.0-os]

Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.

[OAuth Core 1.0]

Hammer-Lahav, E., "OAuth Core 1.0 Protocol".

[Appendix A.](#) Client Account and Password Profile Example

In this example, `crm.example.com` is an application server that has a Protected Resource at `https://crm.example.com/data`. DataDumper is an application acting as a Client that periodically calls `https://crm.example.com/data`. The Protected Resource trusts the Authorization Server `auth.example.net` to determine if a Client has access.

[A.1.](#) Provisioning

The Authorization Server documentation defines the Access Token URL as:

`https://auth.example.net/access_token`

The Authorization Server has defined that the parameter Audience be included in calls to the Access Token URL.

The Client has been provisioned with the following:

Client Account: `datadumper` Client Password: `j2hw7GPs10`

The Protected Resource and the Authorization Server have agreed to

use a Simple Web Token (SWT) for the Access Token with the reserved attributes Issuer, Audience, ExpiresOn and the public attribute net.example.auth.account and have exchanged the following HMAC key value (expressed in base 64):

```
3iK5ZYAoBQuOqSgF/YqlDw70HKRmbyXkrl5f4SJ4Toc=
```

A.2. Client Requests Access Token

The Client makes an HTTPS POST to:

```
https://auth.example.net/access_token
```

With the following message body:

```
wrap_name=datadumper&wrap_password=j2hw7GPsl0&Audience=crm.example.com
```

A.3. Successful Access Token Response from Authorization Server

The Authorization Server checks that the Client Password j2hw7GPsl0 is associated with the Client Name datadumper and that the Client is authorized to access crm.example.com. The Authorization Server notes the time is 2010-02-03T04:05:06Z, which is 1265198706 seconds since 1970-01-01T0:0:0Z. The Authorization Server would like the Access Token to expire in an hour, so 3600 is added to the current time. The Authorization Server then uses the values:

```
net.example.auth.account:  
datadumper ExpiresOn: 1265202306 (1265198706 + 3600)  
Audience: crm.example.com  
Issuer: auth.example.net
```

and the agreed HMAC key to generate the following SWT:

```
net.example.auth.account=datadumper&ExpiresOn=1265202306&Audience=crm.  
example.com&Issuer=auth.example.net&HMACSHA256=N9%2F%2F0tSos78Me36%2Bi  
oBH0sFKfd7eCsURlEIheoUbCJk%3D
```

The Authorization Server then responds to the Clients HTTPS request with:

```
HTTP 200 OK
```

and the Access Token and lifetime of the Access Token as application/x-www-form-urlencoded data in the body of the message as such:

```
wrap_access_token=net.example.auth.account%3Ddatadumper%26ExpiresOn%3D1265202306%26Audience%3Dcrm.example.com%26Issuer%3Dauth.example.net%26HMACSHA256%3DN9%252F%252F0tSos78Me36%252BioBH0sFKfd7eCsURlEIheoUbcJk%253D&wrap_access_token_expires_in=3600
```

A.4. Client Calls Protected Resource

The Client now has an Access Token valid for an hour. The Client makes an API call to:

```
https://crm.example.com/data
```

including the following HTTP header:

```
Authorization: WRAP access_token="net.example.auth.account=datadumper&ExpiresOn=1265202306&Audience=crm.example.com&Issuer=auth.example.net&HMACSHA256=N9%2F%2F0tSos78Me36%2BioBH0sFKfd7eCsURlEIheoUbcJk%3D"
```

The Protected Resources verifies the SWT and performs the Client's request per the authorization attributes in the SWT.

Appendix B. Web App Profile Example

In this example, Jane, the User, listens to music from music.example.com and updates her status at status.example.com. When listening to music, Jane would like her status to be updated at the start of each song. From an OAuth WRAP perspective, the Client is music.example.com, the Protected Resource is https://status.example.com/update, and auth.example.com is the Authorization Server trusted by status.example.com.

B.1. Provisioning

The Authorization Server documentation defines the following URLs:

```
User Authorization URL: https://auth.example.com/user_authorization
Access Token URL:      https://auth.example.com/access_token
Refresh Token URL:     https://auth.example.com/refresh_token
```

The Authorization Server has defined that if the Client wants authorization to update a User's status, that the Client include the wrap_scope parameter with the value status_update when requesting authorization.

The Client has been provisioned with:

Client Identifier: music.example.com

Client Secret: 7F2986DF2342914A

The Client has registered the Callback URL:

https://music.example.com/auth_callback

The Protected Resource and the Authorization Server have agreed to use a Simple Web Token (SWT) for the Access Token with the reserved attributes Issuer, Audience, ExpiresOn and the public attributes com.example.auth.account, com.example.auth.client and com.example.auth.scope. They have exchanged the following HMAC key value (expressed in base 64):

Zt9JlLlQvPYRSCK9PgSjrxRUBWe7lbEYsZCdM+sJCF4=

B.2. Client Directs the User to the Server

Jane informs music.example.com that she would like her status at status.example.com to be updated when a new song starts playing. The music.example.com website maintains user sessions with a URL parameter named session which has the value Vn3IG2FRALSEQX2Nxr at this time for Jane. The Client will use wrap_client_state to maintain the session value. The Client redirects Jane's browser to the Authorization Server's User Authorization URL appending parameters for the Client Identifier, Callback URL, Client state and authorization scope.

https://auth.example.com/user_authorization?wrap_client_id=music.example.com&wrap_callback=http%3A%2F%2Fmusic.example.com%2Fauth_callback&wrap_client_state=Vn3IG2FRALSEQX2Nxr&wrap_scope=status_update

B.3. Authorization Server Confirms Delegation Request with User

The Authorization Server verifies the supplied Client Identifier music.example.com has been registered and has the Callback URL https://music.example.com/auth_callback. The Authorization Server authenticates that the User it is dealing with is Jane, and then asks Jane to authorize music.example.com to update Jane's status at status.example.com. Jane approves the request and the Authorization Server generates a Verification Code with the value 46YEXQjVit6T3nQ8, stores it with the Client Identifier, Callback URL and the current time.

B.4. Server Directs User back to the Client

The Server redirects Jane back to the Client's Callback URL with the Verification Code and Client State appended:

`https://music.example.com/auth_callback?wrap_verification_code=46YEXQj
Vit6T3nQ8&wrap_client_state=Vn3IG2FRALSEQX2Nxr`

B.5. Client Requests Access Token

The Client makes an HTTPS POST request to:

`https://auth.example.com/access_token`

With the following message body:

`wrap_client_id=music.example.com&wrap_client_secret=7F2986DF2342914A&w
rap_verification_code=46YEXQjVit6T3nQ8&wrap_callback=http%3A%2F%2Fmusi
c.example.com%2Fauth_callback`

B.6. Successful Access Token Response from Authorization Server

The Authorization Server verifies that the Verification Code is still valid, has not been used, and is associated with the Client ID, Client Secret and Callback URL Password. The Authorization Server then generates a Refresh Token with the value:

`MfdWTc+v9MXhpc+d/csrKFMPfj1RySm6CzIjmTBGN6w=`

The Authorization Server notes the time is 2010-01-02T03:04:05Z, which is 1262430245 seconds since 1970-01-01T0:0:0Z. The Authorization Server then uses the values:

`com.example.auth.scope: status_update
com.example.auth.account: Jane
com.example.auth.client: music.example.com
ExpiresOn: 1262433845 (1262430245 + 3600 seconds later)
Audience: status.example.com
Issuer: auth.example.com`

and the agreed HMAC key to generate the following SWT:

`com.example.auth.scope=status_update&com.example.auth.account=Jane&com
.example.auth.client=music.example.com&ExpiresOn=1262433845&Audience=s
tatus.example.com&Issuer=auth.example.com&HMACSHA256=3xZAYzJrtYCQgkAF3
iqElp1DhyKkPhq947j04NcDocQ%3D`

The Authorization Server then responds to the Clients HTTPS request with:

`HTTP 200 OK`

and the Refresh Token, Access Token and lifetime of the Access Token

as application/x-www-form-urlencoded data in the body of the message as such:

```
wrap_refresh_token=MfdWTc%2Bv9MXhpc%2Bd%2FcSrKFMPfj1RySm6CzIjmTBGN6w%3D&wrap_access_token=com.example.auth.scope%3Dstatus_update%26com.example.auth.account%3DJane%26com.example.auth.client%3Dmusic.example.com%26ExpiresOn%3D1262433845%26Audience%3Dstatus.example.com%26Issuer%3Dauth.example.com%26HMACSHA256%3D3xZAYzJRtYCQgkAF3iqElp1DhyKkPhq947j04NcDocQ%253D&wrap_access_token_expires_in=3600
```

The Client now has a Refresh Token and Access Token valid for an hour. The Client stores the Refresh Token for later use.

B.7. Client Calls Protected Resource

A few minutes later, music.example.com starts playing a new song for Jane. The Client updates Jane's status at status.example.com by making an API call to:

```
https://status.example.com/update
```

including the following HTTP header:

```
Authorization: WRAP access_token="com.example.auth.scope=status_update&com.example.auth.account=Jane&com.example.auth.client=music.example.com&ExpiresOn=1262433845&Audience=status.example.com&Issuer=auth.example.com&HMACSHA256=3xZAYzJRtYCQgkAF3iqElp1DhyKkPhq947j04NcDocQ%3D"
```

The Protected Resources verifies the SWT, confirms the authorization contained in the SWT, and updates Jane's status.

B.8. Client Refreshes Access Token

An hour passes by and music.example.com starts playing another new song for Jane. The Client again makes an API call to status.example.com including the same HTTP Authorization header. Unlike previous calls where the status update was performed, the Protected Resource returns the following error response:

```
HTTP 401 Unauthorized
```

and the HTTP header:

```
WWW-Authenticate: WRAP
```

The Client determines it probably needs a new Access Token, retrieves the Refresh Token and makes an HTTPS POST to:

`https://auth.example.com/refresh_token`

including the Client Identifier, Client Secret and Refresh Token in the message body as:

```
wrap_client_id=music.example.com&wrap_client_secret=7F2986DF2342914A&wrap_refresh_token=MfdWTc%2Bv9MXhpc%2Bd%2FcSrKFMPfj1RySm6CzIjmTBGN6w%3D
```

The Authorization Server looks up the data associated with the Refresh Token, determines `music.example.com` is still authorized to update Jane's status, and determines it will generate a new Access Token for the Client that expires in an hour. The time is now `2010-01-02T04:15:23Z`, which results in an Access Token expiry time of `1262438123` seconds since `1970-01-01T0:0:0Z`. The Authorization Server generates a new Access Token and returns it in the body of the message as:

```
wrap_access_token=com.example.auth.scope=status_update&com.example.auth.account=Jane&com.example.auth.client=music.example.com&ExpiresOn=1262438123&Audience=status.example.com&Issuer=auth.example.com&HMACSHA256=AT4TFChHgylItEWAjK7MFRJuvUS3WLVzO%2F68gvIRQI%3D&wrap_access_token_expires_in=3600
```

The Client takes the new Access Token and uses it to successfully update Jane's status at `status.example.com`.

Authors' Addresses

Dick Hardt (editor)
Microsoft

Email: `dick.hardt@gmail.com`

Allen Tom
Yahoo!

Email: `atom@yahoo-inc.com`

Brian Eaton
Google

Email: `beaton@google.com`

Yaron Goland
Microsoft

Email: yarong@microsoft.com