Network Working Group	N. Sakimura	
Internet-Draft	Nomura Research Institute	
Intended status: Standards Track	June 17, 2010	
Expires: December 19, 2010		

TOC

Request by Reference ver.1.0 for OAuth 2.0
draft-sakimura-oauth-requrl-00

Abstract

This document defines a simple mechanism of making request by reference in OAuth 2.0. The reference is given as URL and request parameters are defined as JSON object which is pointed by the URL.

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

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
This Internet-Draft will expire on December 19, 2010.

Copyright Notice

Table of Contents

---

1. Introduction

TOC

An additional parameter 'request_url' is introduced as one of the request parameter of the flows of OAuth 2.0 (Hammer-Lahav, E., “The OAuth 2.0 Protocol,” June 2010.) [oauth2].
There are a few cases that this is useful such as:

1. When the User Agent looks like it would not like long URLs - It is entirely possible that some extension makes a long URL. For example, the client might want to send a public key with the request.

2. When the server wants the request non-repudiation - The server might want the request to be non-repudiable. It is possible to sign the request dynamically, but a simpler way of doing it is to make a signed request file and put it on the client. It can just be done by a client utility or something, so that the private key does not even have to reside on the server nor the client needs to program anything.

3. When the server wants the requests to be cacheable - If the request_url comes with sha256 hash as defined in FIPS180-2 (U.S. Department of Commerce and National Institute of Standards and Technology, “Secure Hash Signature Standard,” .) [FIPS180-2]of the file, the server knows if the file has changed without fetching it, so it does not have to re-fetch a same file, which is a win as well.

4. When the client wants to simplify the implementation without compromising the security If the request parameters go through the Browser, it may be tampered at the browser even if TLS was used. This implies we need to have signature on the request as well. However, if HTTPS request_url was used, it is not going to be tampered, thus we now do not have to sign the request. This simplifies the implementation.

---

2. Terminology

TOC

Following parameter is defined as a request and response parameter.

request_url The absolute URL from which request parameters are obtained.

Request File This is a physical or logical file that the 'request_url' points to. It is in JSON format. It MAY include all the potential variables including extension and non-oauth variables. Request File can optionally be digitally signed. To sign the request file, magic signatures (Panzer, J. and B. Laurie, “Magic Signatures,” February 2010.) [magic_signatures] is used.

---

3. Formats

TOC

There are several format that this document defines.

3.1. JSON OAuth Request

TOC

The OAuth authorization request parameters are included in the entity body of the HTTP response using the "application/json" media type as defined by JSON (Crockford, D., “The application/json Media Type for

JavaScript Object Notation (JSON),” July 2006.) [RFC4627]. It MAY include any other parameters as well. The parameters are serialized into a JSON structure by adding each parameter at the highest structure level. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers.
For example:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "redirect_url":"https://example.com/rp/endpoint_url",
    "cliend_id":"http://example.com/rp/",
    "state":"ab890f0d"
}
```

3.2. Signed JSON OAuth Request

TOC

To achieve non-repudiation and ascertained provenance of the JSON OAuth Request when it was obtained through unprotected channel, one MAY use asymmetric signature as in [magic_signatures] (Panzer, J. and B. Laurie, “Magic Signatures,” February 2010.) where parameters are as follows.

> "data_type":"application/json"
>
> "encoding":"base64url"
>
> "alg":"RSA-SHA256"
>
> "data":base64url encoded JSON representation of the assertion.

This specification defines an additional parameter.

*certs_url

> Value: A URL from which one can retrieve PEM format X.509 certificate. It is used as the replacement of "keyhash". Only either of the keyhash or certs_url MAY be used. When certs_url was used, the processor MUST obtain the certificate from this URL and use public key contained in it to verify the signature. In addition, the processor MUST verify the domain of the "request_url" against the CN of the certificate.

Any other variables can be included in any level of the JSON structure. The processor MUST ignore the parameter that it does not understand.

Following is the non-normative illustration of a signed response. (Note: Line wraps in the values are only for the display purpose. New lines MUST be escaped in JSON values.)

```
{
  "data_type":"application/json",
  "encoding":"base64url",
  "alg":"RSA-SHA256",
  "data":"base64url encoded data without padding",
  "sigs": [
    {
    "value": "EvGSD2vi8qYcveHnb-rrlok07qnCXjn8YSeCDDXlbhILSabgvNsPpbe76
    up8w63i2fWHvLKJzeGLKfyHg8ZomQ",
    "certs_url": "https://rp.example.com/certs.pem"
    }
  ]
}
```

3.3. Encrypted Format

TOC

The HTTP response may be encrypted by the receiving party's public key for audience restriction and confidentiality. If it is encrypted, the data is formatted as JSON Encryption Envelope (Bradeley, J. and N. Sakimura, "JSON Encryption Envelope," February 2010.) [json_enc]. The following parameter MUST be set as follows:

*data_type - "http://openid.net/specs/ab/1.0#openid2json-enc"

Following is a non-normative example of encrypted payload.

```
{
    "class_id":"http://jsonenc.info/json-encryption/1.0/",
    "data_type":"http://openid.net/specs/ab/1.0#openid2json-enc",
    "enc_data":"b5guwzFgvrIUd7XcXI0bAFrg-....O69VKhY",
    "enc_type_asy":"http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p",
    "enc_type":"http://www.w3.org/2001/04/xmlenc#aes256-cbc",
    "enc_key":"mHM2ongmZlPVexe....2lsBNdw",
    "enc_iv":"_b4INfYIRwLPZdxB2L7wJg",
    "enc_ref":"https://rp.example.com/rpf_ax.json",
    "enc_thumbprint":"511e7a9cfe5eda16fa70f553c2dfa3c473e06423"
}
```

TOC

4. Obtaining End-User Authorization

When the client is obtaining End-User Authorization as in Section 3 of OAuth 2.0 (Hammer-Lahav, E., "The OAuth 2.0 Protocol," June 2010.) [oauth2] , the client MAY use request_url instead of any but 'state' parameter. The corresponding Request File MUST have all the necessary parameters which was to be queried in this request. When sending the request_url, the client MAY provide the sha256 hash as defined in FIPS180-2 (U.S. Department of Commerce and National Institute of Standards and Technology, "Secure Hash Signature Standard," .) [FIPS180-2]of the Request File as a fragment to it to assist the cache utilization decision of the Authorization Server.
For example, the client directs the end-user's user-agent to make the following HTTPS request (line breaks are for display purposes only):

```
GET /authorize?request_url=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

The Request File MAY be signed AND/OR encrypted.
Upon receipt of 'requst_url' in the request, the authorization server MUST send a GET request to the 'request_url' to retrieve the content of the Request File unless it is already cached at the Authorization Server.
If the response was signed AND/OR encrypted, it has to be decoded accordingly before being processed.
Then, the Authorization Server MUST reconstruct the complete client request from the original HTTP request and the content of the Request File. Then, the process continues as described in Section 3 of OAuth 2.0 (Hammer-Lahav, E., "The OAuth 2.0 Protocol," June 2010.) [oauth2] .

4.1. Authorization Server Response

TOC

Authorization Server Response is created and sent to the client as in Section 3.1 of OAuth 2.0 (Hammer-Lahav, E., "The OAuth 2.0 Protocol," June 2010.) [oauth2] .
In addition, this document defines additional 'error' values as follows:

*"invalid_request_url" - The provided request_url was not available.

*"invalid_request_file_format" - The Request File format was invalid.

*"invalid_request_params" - The parameter set provided in the Request File was invalid.

5. IANA Considerations

TOC

This document makes no request of IANA.

6. Security Considerations

TOC

When obtaining the Request FIle, the Authorization Server SHOULD use either HTTP over TLS 1.2 as defined in RFC5246 (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.) [RFC5246] AND/OR Signed JSON Request.
If the Request File contains personally identifiable or sensitive information, the "request_url" MUST be of one-time use and MUST have large enough entropy deemed necessary with applicable security policy.
For higher security requirement, using Encryption described in (Encrypted Format) is strongly recommended.
[[ToDo]]

7. Acknowledgements

TOC

Following people contributed to creating this document through the OpenID Artifact Binding 1.0 (openid-specs-ab@openid.net, “OpenID Artifact Binding 1.0,” June 2010.) [openid_ab] .
Breno de Medeiros (Google), Hideki Nara (TACT), John Bradley (Wingaa) <author>, Nat Sakimura (NRI) <author/editor>, Ryo Itou (Yahoo! Japan)
Many people joined the discussion of the above at IIW and other occasions including Allen Tom (Yahoo!), George Fletcher (AOL), Dick Hardt (Independent) [[ToDo]]
In addition following people contributed to this and previous versions through The OAuth Working Group.
David Recordon (Facebook), Luke Shepard (Facebook), James H. Manger (Telstra), Marius Scurtescu (Google), John Panzer (Google), Dirk Balfanz (Google).

8. References

TOC

8.1. Normative References

TOC

[FIPS180-2]	U.S. Department of Commerce and National Institute of Standards and Technology, “Secure Hash Signature Standard,” FIPS 180-2. Defines Secure Hash Algorithm 256 (SHA256)
[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC4627]	Crockford, D., “The application/json Media Type for JavaScript Object Notation (JSON),” RFC 4627, July 2006 (TXT).
[RFC5246]	Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” RFC 5246, August 2008 (TXT).
[json_enc]	Bradeley, J. and N. Sakimura, “JSON Encryption Envelope,” February 2010.
[magic_signatures]	Panzer, J. and B. Laurie, “Magic Signatures,” February 2010.
[oauth2]	Hammer-Lahav, E., “The OAuth 2.0 Protocol,” June 2010.

8.2. Informative References

TOC

[openid_ab]	openid-specs-ab@openid.net, “OpenID Artifact Binding 1.0,” June 2010.

Author's Address

TOC

	Nat Sakimura
	Nomura Research Institute
	1-6-5 Marunouchi, Marunouchi Kitaguchi Bldg.
	Chiyoda-ku, Tokyo 100-0005
	Japan
Phone:	+81-3-5533-2111
Email:	n-sakimura@nri.co.jp