Internet Engineering Task Force	N. Sakimura, Ed.
Internet-Draft	Nomura Research Institute
Intended status: Standards Track	J. Bradley
Expires: April 29, 2012	independent
	October 27, 2011

Request by JSON ver.1.0 for OAuth 2.0
draft-sakimura-oauth-requrl-01

Abstract

The authorization request in OAuth 2.0 utilizes query parameter serizalization. This specification defines the authorization request using JWT serialization. The request is sent thorugh 'request' parameter or by reference through 'request_uri' that points to the JWT.

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 April 29, 2012.

Copyright Notice

Table of Contents

1. Introduction

The parameters 'request' and 'request_url' are introduced as additional authorization request parameters for the OAuth 2.0 *[oauth2]* flows. The 'request' parameter is a JSON Web Token (JWT) [JWT] whose body holds the JSON encoded OAuth 2.0 authorization request parameters. The [JWT] can be passed to the authorization endpoint by reference, in which case the parameter 'request_uri' is used instead of the 'request'.
Using [JWT] as the request encoding instead of query parameters has several advantages:

1. The request may be signed so that integrity check may be implemented. If a suitable algorithm is used for the signing, then non-repudiation property may be obtained in addition.

2. The request may be encrypted so that end-to-end confidentiality may be obtained even if in the case TLS connection is terminated at a gateway or similar device.

There are a few cases that request by reference is useful such as:

1. When it is detected that the User Agent dose't suport long URLs - It is entirely possible that some extensions may extend the URL. For example, the client might want to send a public key with the request.

2. Static signature: The client may make a signed request file and put it on the client. This can just be done by a client utility

or other process, so that the private key does not have to reside on the client, simplifying programming.

3. When the server wants the requests to be cacheable - The request_uri can include a sha256 hash of the file, as defined in FIPS180-2 *[FIPS180-2]*, 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, they may be tampered in 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.

1.1. 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 *[RFC2119]*.

2. Terminology

Following parameter is defined as a request and response parameter.

request object A [JWT] that holds OAuth 2.0 authorization requests as JSON object in its body. It MAY include all the potential variables including extension and non-oauth variables. Request object can optionally be digitally signed or signed and encrypted. To sign, [JWS] is used. To encrypt, [JWT] is used.

request_uri The absolute URL from which the request object is obtained.

Request File This is a physical or logical file that the 'request_url' points to.

3. Authorization Request Object

The authorization request object is are included as the top level members of JSON *[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.
The Authorization Request object is used to provide authorization request parameters. It MUST contain all REQUIRED OAuth 2.0 authorization request parameters and MAY contain optional and extension

parameters. It is a JSON Web Token (JWT) *[JWT]* that has the JSON object that holds the OAuth 2.0 authorization request parameters. The parameters are included as the top level members of JSON *[RFC4627]*. Parameter names and string values are included as JSON strings. Numerical values are included as JSON numbers. It MAY include any extension parameters. This JSON *[RFC4627]* constitues the body of the [JWT].
The [JWT] MAY be signed or unsigned. When it is unsigned, it will be indicated by the [JWT] "signed":"none" convention in the [JWT] header. If signed, the authorization request object SHOULD contain the standard [JWT] "iss" and "aud" claims.
Following is the example of the JSON which consitutes the body of the [JWT].

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

The following is a non-normative example of a [JWT] encoded authorization request object. It includes extension variables such as "nonce", "userinfo", and "id_token". Note that the line wraps within the values are for display purpose only:

```
JWT algorithm = HS256
HMAC HASH Key = 'aaa'

JSON Encoded Header = "{"alg":"HS256","typ":"JWT"}"
JSON Encoded Payload = "{"response_type":"code id_token",
    "client_id":"s6BhdRkqt3",
    "redirect_uri":"https://client.example.com/cb",
    "scope":"openid profile",
    "state":"af0ifjsldkj",
    "nonce":"n-0S6_WzA2Mj",
    "userinfo":{"claims":{"name":null,"nickname":{"optional":true},
        "email":null,"verified":null,
        "picture":{"optional":true}},"format":"signed"},
    "id_token":{"max_age":86400,"iso29115":"2"}}"

JWT = eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJyZXNwb25zZV90eXBlIjoiY29kZ
    SBpZF90b2tlbiIsImNsaWVudF9pZCI6InM2QmhkUmtxdDMiLCJyZWRpcmVjdF91cmkiO
    iJodHRwczpcL1wvY2xpZW50LmV4YW1wbGUuY29tXC9jYiIsInNjb3BlIjoib3BlbmlkI
    HByb2ZpbGUiLCJzdGF0ZSI6ImFmMGlmanNsZGtqIiwidXNlcmluZm8iOnsiY2xhaW1zI
    jp7Im5hbWUiOm51bGwsIm5pY2tuYW1lIjp7Im9wdGlvbmFsIjp0cnVlfSwiZW1haWwiO
    m51bGwsInZlcmlmaWVkIjpudWxsLCJwaWN0dXJlIjp7Im9wdGlvbmFsIjp0cnVlfX0sI
    mZvcm1hdCI6InNpZ25lZCJ9LCJpZF90b2tlbiI6eyJtYXhfYWdlIjo4NjQwMCwiaXNvM
    jkxMTUiOiIyIn19.2OiqRgrbrHkA1FZ5p_7bc_RSdTbH-wo_Agk-ZRpD3wY
```

4. Authorization Request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

request REQUIRED unless request_uri is specified. The authorization request object *[aro]* that holds authorization request parameters stated in the section 4 of OAuth 2.0 *[oauth2]*.

request_uri REQUIRED unless request is specified. The absolute URL that points to the authorization request object *[aro]* that holds authorization request parameters stated in the section 4 of OAuth 2.0 *[oauth2]*. When sending the request by request_uri, the client MAY provide the sha256 hash as defined in FIPS180-2 *[FIPS180-2]*of the Request File as the fragment to it to assist the cache utilization decision of the Authorization Server.

state RECOMMENDED. An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter SHOULD be used for preventing cross-site request forgery as described in Section 10.12. of OAuth 2.0 *[oauth2]*

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.
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_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

The autorization request object MAY be signed AND/OR encrypted.
Upon receipt of request_uri in the request, the authorization server MUST send a GET request to the request_uri to retrieve the authorization request object 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 object. Then, the process continues as described in Section 3 of OAuth 2.0 *[oauth2]* .

5. Authorization Server Response

Authorization Server Response is created and sent to the client as in Section 4 of OAuth 2.0 *[oauth2]* .

In addition, this document defines additional 'error' values as follows:

*"invalid_request_uri" - The provided request_uri was not available.

*"invalid_request_format" - The Request Object format was invalid.

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

6. IANA Considerations

This document registers following error strings to the OAuth Error Registry.

*"invalid_request_uri" - The provided request_uri was not available.

*"invalid_request_format" - The Request Object format was invalid.

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

7. Security Considerations

In addition to the all the security considerations discussed in OAuth 2.0 *[oauth2]*, the following security considerations SHOULD be taken into account.
When sending the authorization request object through request parameter, it SHOULD be signed with [JWS].
When obtaining the Request FIle, the Authorization Server SHOULD use either HTTP over TLS 1.2 as defined in RFC5246 *[RFC5246]* AND/OR [JWS].
If the request object contains personally identifiable or sensitive information, the "request_uri" MUST be of one-time use and MUST have large enough entropy deemed necessary with applicable security policy.
For higher security requirement, using [JWE] is strongly recommended.
[[ToDo]]

8. Acknowledgements

Following people contributed to creating this document through the OpenID Connect 1.0 *[openid_ab]* .
Breno de Medeiros (Google), Hideki Nara (TACT), John Bradley (Independent) <author>, Nat Sakimura (NRI) <author/editor>, Ryo Itou (Yahoo! Japan), George Fletcher (AOL), Justin Richer (Mitre), Edmund Jay (MGI1), (add yourself).
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), (add yourself).

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.
[RFC4627]	Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
[RFC5246]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
[FIPS180-2]	U.S. Department of CommerceNational Institute of Standards and Technology, "Secure Hash Signature Standard", FIPS 180-2, August 2002. Defines Secure Hash Algorithm 256 (SHA256)
[oauth2]	Hammer-Lahav, E., "The OAuth 2.0 Protocol", June 2010.
[JWT]	Jones, M.B., Balfanz, D., Bradley, J., Goland, Y.Y., Panzer, J., Sakimura, N. and P. Tarjan, "JSON Web Token", July 2011.
[JWS]	Jones, M.B., Balfanz, D., Bradley, J., Goland, Y.Y., Panzer, J., Sakimura, N. and P. Tarjan, "JSON Web Signature (JWS)", April 2011.
[JWE]	Jones, M.B., Bradley, J. and N. Sakimura, "JSON Web Encryption (JWE)", March 2011.

9.2. Informative References

[openid_ab]	openid-specs-ab@openid.net, , "OpenID Connect 1.0", October 2011.

Authors' Addresses

Nat Sakimura editor 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

John Bradley Bradley independent EMail: ve7jtb@ve7jtb.com