OAuth Working Group N. Sakimura
Internet-Draft Nomura Research Institute
Intended status: Standards Track J. Bradley
Expires: April 24, 2019 Yubico
October 21, 2018

The OAuth 2.0 Authorization Framework: JWT Secured Authorization Request (JAR)

draft-ietf-oauth-jwsreq-17

Abstract

The authorization request in OAuth 2.0 described in RFC 6749 utilizes query parameter serialization, which means that Authorization Request parameters are encoded in the URI of the request and sent through user agents such as web browsers. While it is easy to implement, it means that (a) the communication through the user agents are not integrity protected and thus the parameters can be tainted, and (b) the source of the communication is not authenticated. Because of these weaknesses, several attacks to the protocol have now been put forward.

This document introduces the ability to send request parameters in a JSON Web Token (JWT) instead, which allows the request to be signed with JSON Web Signature (JWS) and encrypted with JSON Web Encryption (JWE) so that the integrity, source authentication and confidentiality property of the Authorization Request is attained. The request can be sent by value or by reference.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Authorization Request in OAuth 2.0 [RFC6749] utilizes query parameter serialization and is typically sent through user agents such as web browsers.

For example, the parameters "response_type", "client_id", "state", and "redirect_uri" are encoded in the URI of the request:

```
GET /authorize?response_type=code&client_id=s6BhdRkqt3&state=xyz
&redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

While it is easy to implement, the encoding in the URI does not allow application layer security with confidentiality and integrity protection to be used. While TLS is used to offer communication security between the Client and the user-agent as well as the user-agent and the Authorization Server, TLS sessions are terminated in the user-agent. In addition, TLS sessions may be terminated prematurely at some middlebox (such as a load balancer).

As the result, the Authorization Request of [RFC6749] has shortcomings in that:

(a) the communication through the user agents are not integrity protected and thus the parameters can be tainted (integrity protection failure)

(b) the source of the communication is not authenticated (source authentication failure)

(c) the communication through the user agents can be monitored (containment / confidentiality failure).

Due to these inherent weaknesses, several attacks against the protocol, such as Redirection URI rewriting and Mix-up attack [FETT], have been identified.

The use of application layer security mitigates these issues.

The use of application layer security allows requests to be prepared by a third party so that a client application cannot request more permissions than previously agreed. This offers an additional degree of privacy protection.

Furthermore, the request by reference allows the reduction of over-the-wire overhead.

The JWT [RFC7519] encoding has been chosen because of

(1) its close relationship with JSON, which is used as OAuth's response format

(2) its developer friendliness due to its textual nature

(3) its relative compactness compared to XML

(4) its development status that it is an RFC and so is its associated signing and encryption methods as [RFC7515] and [RFC7516]

(5) the relative ease of JWS and JWE compared to XML Signature and Encryption.

The parameters "request" and "request_uri" are introduced as additional authorization request parameters for the OAuth 2.0 [RFC6749] flows. The "request" parameter is a JSON Web Token (JWT) [RFC7519] whose JWT Claims Set holds the JSON encoded OAuth 2.0 authorization request parameters. This JWT is integrity protected and source authenticated using JWS.

The JWT [RFC7519] can be passed to the authorization endpoint by reference, in which case the parameter "request_uri" is used instead of the "request".

Using JWT [RFC7519] as the request encoding instead of query parameters has several advantages:

(a) (integrity protection) The request can be signed so that the integrity of the request can be checked.

(b) (source authentication) The request can be signed so that the signer can be authenticated.

(c) (confidentiality protection) The request can be encrypted so that end-to-end confidentiality can be provided even if the TLS connection is terminated at one point or another.

(d) (collection minimization) The request can be signed by a third party attesting that the authorization request is compliant with a certain policy. For example, a request can be pre-examined by a third party that all the personal data requested is strictly necessary to perform the process that the end-user asked for, and statically signed by that third party. The authorization server then examines the signature and shows the conformance status to the end-user, who would have some assurance as to the legitimacy of the request when authorizing it. In some cases, it may even be desirable to skip the authorization dialogue under such circumstances.

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

1. When it is desirable to reduce the size of transmitted request. The use of application layer security increases the size of the request, particularly when public key cryptography is used.

2. When the client does not want to do the crypto. The Authorization Server may provide an endpoint to accept the Authorization Request through direct communication with the Client so that the Client is authenticated and the channel is TLS protected.

This capability is in use by OpenID Connect [OpenID.Core].

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

For the purposes of this specification, the following terms and definitions in addition to what is defined in OAuth 2.0 Framework [RFC6749], JSON Web Signature [RFC7515], and JSON Web Encryption [RFC7519] apply.

2.1. Request Object

JWT [RFC7519] that holds an OAuth 2.0 authorization request as JWT Claims Set

2.2. Request Object URI

Absolute URI from which the Request Object (Section 2.1) can be obtained

3. Symbols and abbreviated terms

The following abbreviations are common to this specification.

JSON Javascript Object Notation

JWT JSON Web Token

JWS JSON Web Signature

JWE JSON Web Encryption

URI Uniform Resource Identifier

URL Uniform Resource Locator

4. Request Object

A Request Object (Section 2.1) is used to provide authorization request parameters for an OAuth 2.0 authorization request. It MUST contains all the OAuth 2.0 [RFC6749] authorization request parameters including extension parameters. The parameters are represented as the JWT claims. Parameter names and string values MUST be included as JSON strings. Since Request Objects are handled across domains and potentially outside of a closed ecosystem, per section 8.1 of [RFC8259], these JSON strings MUST be encoded using UTF-8 [RFC3629]. Numerical values MUST be included as JSON numbers. It MAY include any extension parameters. This JSON [RFC7159] constitutes the JWT Claims Set defined in JWT [RFC7519]. The JWT Claims Set is then signed or signed and encrypted.

To sign, JSON Web Signature (JWS) [RFC7515] is used. The result is a JWS signed JWT [RFC7519]. If signed, the Authorization Request Object SHOULD contain the Claims "iss" (issuer) and "aud" (audience) as members, with their semantics being the same as defined in the JWT [RFC7519] specification.

To encrypt, JWE [RFC7516] is used. When both signature and encryption are being applied, the JWT MUST be signed then encrypted as advised in the section 11.2 of [RFC7519]. The result is a Nested JWT, as defined in [RFC7519].

The Authorization Request Object MAY be sent by value as described in Section 5.1 or by reference as described in Section 5.2.

"request" and "request_uri" parameters MUST NOT be included in Request Objects.

The following is an example of the Claims in a Request Object before base64url encoding and signing. Note that it includes extension variables such as "nonce" and "max_age".

```
{
 "iss": "s6BhdRkqt3",
 "aud": "https://server.example.com",
 "response_type": "code id_token",
 "client_id": "s6BhdRkqt3",
 "redirect_uri": "https://client.example.org/cb",
 "scope": "openid",
 "state": "af0ifjsldkj",
 "nonce": "n-0S6_WzA2Mj",
 "max_age": 86400
}
```

Signing it with the "RS256" algorithm results in this Request Object value (with line wraps within values for display purposes only):

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3MiOiAiczZCaGRSa3
F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsDQogInJl
c3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsDQogImNsaWVudF9pZCI6ICJzNk
JoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8vY2xpZW50LmV4YW1w
bGUub3JnL2NiIiwNCiAic2NvcGUiOiAib3BlbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
Zqc2xka2oiLA0KICJub25jZSI6ICJuLTBTNl9XekEyTWoiLA0KICJtYXhfYWdlIjog
ODY0MDAsDQogImNsYWltcyI6IA0KICB7DQogICAidXNlcmluZm8iOiANCiAgICB7DQ
ogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgIm5p
Y2tuYW1lIjogbnVsbCwNCiAgICAgImVtYWlsIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
wNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAg
ICAgInBpY3R1cmUiOiBudWxsDQogICAgfSwNCiAgICJpZF90b2tlbiI6IA0KICAgIH
sNCiAgICAgImdlbmRlciI6IG51bGwsDQogICAgICJiaXJ0aGRhdGUiOiB7ImVzc2Vu
dGlhbCI6IHRydWV9LA0KICAgICAiYWNyIjogeyJ2YWx1ZXMiOiBbInVybjptYWNlOm
luY29tbW9uOmlhcDpzaWx2ZXIiXX0NCiAgICB9DQogIH0NCn0.nwwnNsk1-Zkbmnvs
F6zTHm8CHERFMGQPhos-EJcaH4Hh-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyF
KzuMXZFSZ3p6Mb8dkxtVyjoy2GIzvuJT_u7PkY2t8QU9hjBcHs68PkgjDVTrG1uRTx
0GxFbuPbj96tVuj11pTnmFCUR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8K
ol-cSLWoYE9l5QqholImzjT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPG
iyon_-Te111V8uE83IlzCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

The following RSA public key, represented in JWK format, can be used to validate the Request Object signature in this and subsequent Request Object examples (with line wraps within values for display purposes only):

```
 {
  "kty":"RSA",
  "kid":"k2bdc",
  "n":"y9Lqv4fCp6Ei-u2-ZCKq83YvbFEk6JMs_pSj76eMkddWRuWX2aBKGHAtKlE5P
       7_vn__PCKZWePt3vGkB6ePgzAFu08NmKemwE5bQI0e6kIChtt_6KzT5OaaXDF
       I6qCLJmk51Cc4VYFaxgqevMncYrzaW_50mZ1yGSFIQzLYP8bijAHGVjdEFgZa
       ZEN9lsn_GdWLaJpHrB3ROlS50E45wxrlg9xMncVb8qDPuXZarvghLL0HzOuYR
       adBJVoWZowDNTpKpk2RklZ7QaBO7XDv3uR7s_sf2g-bAjSYxYUGsqkNA9b3xV
       W53am_UZZ3tZbFTIh557JICWKHlWj5uzeJXaw",
  "e":"AQAB"
 }
```

5. Authorization Request

The client constructs the authorization request URI by adding one of the following parameters but not both to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

request The Request Object (Section 2.1) that holds authorization request parameters stated in section 4 of OAuth 2.0 [RFC6749].

request_uri The absolute URI as defined by RFC3986 [RFC3986] that points to the Request Object (Section 2.1) that holds authorization request parameters stated in section 4 of OAuth 2.0 [RFC6749].

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:

```
GET /authz?request=eyJhbG..AlMGzw HTTP/1.1
Host: server.example.com
```

The value for the request parameter is abbreviated for brevity.

The authorization request object MUST be one of the following:

(a) JWS signed

(b) JWS signed and JWE encrypted

The client MAY send the parameters included in the request object duplicated in the query parameters as well for the backward compatibility etc. However, the authorization server supporting this specification MUST only use the parameters included in the request object.

5.1. Passing a Request Object by Value

The Client sends the Authorization Request as a Request Object to the Authorization Endpoint as the "request" parameter value.

The following is an example of an Authorization Request using the "request" parameter (with line wraps within values for display purposes only):

```
  https://server.example.com/authorize?
    request=eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3MiOiA
    iczZCaGRSa3F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmN
    vbSIsDQogInJlc3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsDQogImNsaWV
    udF9pZCI6ICJzNkJoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8
    vY2xpZW50LmV4YW1wbGUub3JnL2NiIiwNCiAic2NvcGUiOiAib3BlbmlkIiwNCiA
    ic3RhdGUiOiAiYWYwaWZqc2xka2oiLA0KICJub25jZSI6ICJuLTBTNl9XekEyTWo
    iLA0KICJtYXhfYWdlIjogODY0MDAsDQogImNsYWltcyI6IA0KICB7DQogICAidXN
    lcmluZm8iOiANCiAgICB7DQogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWw
    iOiB0cnVlfSwNCiAgICAgIm5pY2tuYW1lIjogbnVsbCwNCiAgICAgImVtYWlsIjo
    geyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJ
    lc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgInBpY3R1cmUiOiBudWxsDQogICAgfSw
    NCiAgICJpZF90b2tlbiI6IA0KICAgIHsNCiAgICAgImdlbmRlciI6IG51bGwsDQo
    gICAgICJiaXJ0aGRhdGUiOiB7ImVzc2VudGlhbCI6IHRydWV9LA0KICAgICAiYWN
    yIjogeyJ2YWx1ZXMiOiBbInVybjptYWNlOmluY29tbW9uOmlhcDpzaWx2ZXIiXX0
    NCiAgICB9DQogIH0NCn0.nwwnNsk1-ZkbmnvsF6zTHm8CHERFMGQPhos-EJcaH4H
    h-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyFKzuMXZFSZ3p6Mb8dkxtVyjoy2
    GIzvuJT_u7PkY2t8QU9hjBcHs68PkgjDVTrG1uRTx0GxFbuPbj96tVuj11pTnmFC
    UR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8Kol-cSLWoYE9l5QqholImz
    jT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPGiyon_-Te111V8uE83Il
    zCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

5.2. Passing a Request Object by Reference

The "request_uri" Authorization Request parameter enables OAuth authorization requests to be passed by reference, rather than by value. This parameter is used identically to the "request" parameter, other than that the Request Object value is retrieved from the resource identified by the specified URI rather than passed by value.

The entire Request URI MUST NOT exceed 512 ASCII characters. There are three reasons for this restriction.

1. Many phones in the market as of this writing still do not accept large payloads. The restriction is typically either 512 or 1024 ASCII characters.

2. The maximum URL length supported by older versions of Internet Explorer is 2083 ASCII characters.

3. On a slow connection such as 2G mobile connection, a large URL would cause the slow response and therefore the use of such is not advisable from the user experience point of view.

The contents of the resource referenced by the URI MUST be a Request Object. The "request_uri" value MUST be either URN as defined in RFC8141 [RFC8141] or "https" URI, as defined in 2.7.2 of RFC7230 [RFC7230] . The "request_uri" value MUST be reachable by the Authorization Server.

The following is an example of the contents of a Request Object resource that can be referenced by a "request_uri" (with line wraps within values for display purposes only):

```
  eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3MiOiAiczZCaGRSa3
  F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsDQogInJl
  c3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsDQogImNsaWVudF9pZCI6ICJzNk
  JoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8vY2xpZW50LmV4YW1w
  bGUub3JnL2NiIiwNCiAic2NvcGUiOiAib3BlbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
  Zqc2xka2oiLA0KICJub25jZSI6ICJuLTBTNl9XekEyTWoiLA0KICJtYXhfYWdlIjog
  ODY0MDAsDQogImNsYWltcyI6IA0KICB7DQogICAidXNlcmluZm8iOiANCiAgICB7DQ
  ogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgIm5p
  Y2tuYW1lIjogbnVsbCwNCiAgICAgImVtYWlsIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
  wNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAg
  ICAgInBpY3R1cmUiOiBudWxsDQogICAgfSwNCiAgICJpZF90b2tlbiI6IA0KICAgIH
  sNCiAgICAgImdlbmRlciI6IG51bGwsDQogICAgICJiaXJ0aGRhdGUiOiB7ImVzc2Vu
  dGlhbCI6IHRydWV9LA0KICAgICAiYWNyIjogeyJ2YWx1ZXMiOiBbInVybjptYWNlOm
  luY29tbW9uOmlhcDpzaWx2ZXIiXX0NCiAgICB9DQogIH0NCn0.nwwnNsk1-Zkbmnvs
  F6zTHm8CHERFMGQPhos-EJcaH4Hh-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyF
  KzuMXZFSZ3p6Mb8dkxtVyjoy2GIzvuJT_u7PkY2t8QU9hjBcHs68PkgjDVTrG1uRTx
  0GxFbuPbj96tVuj11pTnmFCUR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8K
  ol-cSLWoYE9l5QqholImzjT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPG
  iyon_-Te111V8uE83IlzCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

5.2.1. URI Referencing the Request Object

The Client stores the Request Object resource either locally or remotely at a URI the Authorization Server can access. Such facility may be provided by the authorization server or a third party. For example, the authorization server may provide a URL to which the client POSTs the request object and obtains the Requiest URI. This URI is the Request Object URI, "request_uri".

It is possible for the Request Object to include values that are to be revealed only to the Authorization Server. As such, the "request_uri" MUST have appropriate entropy for its lifetime. For the guidance, refer to 5.1.4.2.2 of [RFC6819]. It is RECOMMENDED that it be removed after a reasonable timeout unless access control measures are taken.

The following is an example of a Request Object URI value (with line wraps within values for display purposes only):

```
  https://tfp.example.org/request.jwt#
    GkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
```

5.2.2. Request using the "request_uri" Request Parameter

The Client sends the Authorization Request to the Authorization Endpoint.

The following is an example of an Authorization Request using the "request_uri" parameter (with line wraps within values for display purposes only):

```
  https://server.example.com/authorize?
    response_type=code%20id_token
    &client_id=s6BhdRkqt3
    &request_uri=https%3A%2F%2Ftfp.example.org%2Frequest.jwt
    %23GkurKxf5T0Y-mnPFCHqWOMiZi4VS138cQO_V7PZHAdM
    &state=af0ifjsldkj
```

5.2.3. Authorization Server Fetches Request Object

Upon receipt of the Request, the Authorization Server MUST send an HTTP "GET" request to the "request_uri" to retrieve the referenced Request Object, unless it is stored in a way so that it can retrieve it through other mechanism securely, and parse it to recreate the Authorization Request parameters.

The following is an example of this fetch process:

```
GET /request.jwt HTTP/1.1
Host: tfp.example.org
```

The following is an example of the fetch response:

```
HTTP/1.1 200 OK
Date: Thu, 16 Feb 2017 23:52:39 GMT
Server: Apache/2.2.22 (tfp.example.org)
Content-type: application/jwt
Content-Length: 1250
Last-Modified: Wed, 15 Feb 2017 23:52:32 GMT
```

```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImsyYmRjIn0.ew0KICJpc3MiOiAiczZCaGRSa3
F0MyIsDQogImF1ZCI6ICJodHRwczovL3NlcnZlci5leGFtcGxlLmNvbSIsDQogInJl
c3BvbnNlX3R5cGUiOiAiY29kZSBpZF90b2tlbiIsDQogImNsaWVudF9pZCI6ICJzNk
JoZFJrcXQzIiwNCiAicmVkaXJlY3RfdXJpIjogImh0dHBzOi8vY2xpZW50LmV4YW1w
bGUub3JnL2NiIiwNCiAic2NvcGUiOiAib3BlbmlkIiwNCiAic3RhdGUiOiAiYWYwaW
Zqc2xka2oiLA0KICJub25jZSI6ICJuLTBTNl9XekEyTWoiLA0KICJtYXhfYWdlIjog
ODY0MDAsDQogImNsYWltcyI6IA0KICB7DQogICAidXNlcmluZm8iOiANCiAgICB7DQ
ogICAgICJnaXZlbl9uYW1lIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAgICAgIm5p
Y2tuYW1lIjogbnVsbCwNCiAgICAgImVtYWlsIjogeyJlc3NlbnRpYWwiOiB0cnVlfS
wNCiAgICAgImVtYWlsX3ZlcmlmaWVkIjogeyJlc3NlbnRpYWwiOiB0cnVlfSwNCiAg
ICAgInBpY3R1cmUiOiBudWxsDQogICAgfSwNCiAgICJpZF90b2tlbiI6IA0KICAgIH
sNCiAgICAgImdlbmRlciI6IG51bGwsDQogICAgICJiaXJ0aGRhdGUiOiB7ImVzc2Vu
dGlhbCI6IHRydWV9LA0KICAgICAiYWNyIjogeyJ2YWx1ZXMiOiBbInVybjptYWNlOm
luY29tbW9uOmlhcDpzaWx2ZXIiXX0NCiAgICB9DQogIH0NCn0.nwwnNsk1-Zkbmnvs
F6zTHm8CHERFMGQPhos-EJcaH4Hh-sMgk8ePrGhw_trPYs8KQxsn6R9Emo_wHwajyF
KzuMXZFSZ3p6Mb8dkxtVyjoy2GIzvuJT_u7PkY2t8QU9hjBcHs68PkgjDVTrG1uRTx
0GxFbuPbj96tVuj11pTnmFCUR6IEOXKYr7iGOCRB3btfJhM0_AKQUfqKnRlrRscc8K
ol-cSLWoYE9l5QqholImzjT_cMnNIznW9E7CDyWXTsO70xnB4SkG6pXfLSjLLlxmPG
iyon_-Te111V8uE83IlzCYIb_NMXvtTIVc1jpspnTSD7xMbpL-2QgwUsAlMGzw
```

6. Validating JWT-Based Requests

6.1. Encrypted Request Object

If the request object is encrypted, the Authorization Server MUST decrypt the JWT in accordance with the JSON Web Encryption [RFC7516] specification.

The result is a signed request object.

If decryption fails, the Authorization Server MUST return an "invalid_request_object" error.

6.2. JWS Signed Request Object

The Authorization Server MUST perform the signature validation of the JSON Web Signature [RFC7515] signed request object. For this, the "alg" Header Parameter in its JOSE Header MUST match the value of the pre-registered algorithm. The signature MUST be validated against the appropriate key for that "client_id" and algorithm.

If signature validation fails, the Authorization Server MUST return an "invalid_request_object" error.

6.3. Request Parameter Assembly and Validation

The Authorization Server MUST extract the set of Authorization Request parameters from the Request Object value. The Authorization Server MUST only use the parameters in the Request Object even if the same parameter is provided in the query parameter. The Authorization Server then validates the request as specified in OAuth 2.0 [RFC6749].

If the validation fails, then the Authorization Server MUST return an error as specified in OAuth 2.0 [RFC6749].

7. Authorization Server Response

Authorization Server Response is created and sent to the client as in Section 4 of OAuth 2.0 [RFC6749] .

In addition, this document uses these additional error values:

invalid_request_uri The "request_uri" in the Authorization Request returns an error or contains invalid data.

invalid_request_object The request parameter contains an invalid Request Object.

request_not_supported The Authorization Server does not support the use of the "request" parameter.

request_uri_not_supported The Authorization Server does not support the use of the "request_uri" parameter.

8. TLS Requirements

Client implementations supporting the Request Object URI method MUST support TLS following Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) [BCP195].

To protect against information disclosure and tampering, confidentiality protection MUST be applied using TLS with a cipher suite that provides confidentiality and integrity protection.

HTTP clients MUST also verify the TLS server certificate, using subjectAltName dNSName identities as described in [RFC6125], to avoid man-in-the-middle attacks. The rules and guidelines defined in [RFC6125] apply here, with the following considerations:

- Support for DNS-ID identifier type (that is, the dNSName identity in the subjectAltName extension) is REQUIRED. Certification authorities which issue server certificates MUST support the DNS-ID identifier type, and the DNS-ID identifier type MUST be present in server certificates.

- DNS names in server certificates MAY contain the wildcard character "*".

- Clients MUST NOT use CN-ID identifiers; a CN field may be present in the server certificate's subject name, but MUST NOT be used for authentication within the rules described in [BCP195] .

- SRV-ID and URI-ID as described in Section 6.5 of [RFC6125] MUST NOT be used for comparison.

9. IANA Considerations

This specification requests no actions by IANA.

10. Security Considerations

In addition to the all the security considerations discussed in OAuth 2.0 [RFC6819], the security considerations in [RFC7515], [RFC7516], and [RFC7518] needs to be considered. Also, there are several academic papers such as [BASIN] that provide useful insight into the security properties of protocols like OAuth.

In consideration of the above, this document advises taking the following security considerations into account.

10.1. Choice of Algorithms

When sending the authorization request object through "request" parameter, it MUST either be signed using JWS [RFC7515] or encrypted using JWE [RFC7516] with then considered appropriate algorithm.

10.2. Request Source Authentication

The source of the Authorization Request MUST always be verified. There are several ways to do it in this specification.

(a) Verifying the JWS Signature of the Request Object.

(b) Verifying that the symmetric key for the JWE encryption is the correct one if the JWE is using symmetric encryption.

(c) Verifying the TLS Server Identity of the Request Object URI. In this case, the Authorization Server MUST know out-of-band that the Client uses Request Object URI and only the Client is covered by the TLS certificate. In general, it is not a reliable method.

(d) Authorization Server is providing an endpoint that provides a Request Object URI in exchange for a Request Object. In this case, the Authorization Server MUST perform Client Authentication to accept the Request Object and bind the Client Identifier to the Request Object URI it is providing. Since Request Object URI can be replayed, the lifetime of the Request Object URI MUST be short and preferably one-time use. The entropy of the Request Object URI MUST be sufficiently large. The adequate shortness of the validity and the entropy of the Request Object URI depends on the risk calculation based on the value of the resource being protected. A general guidance for the validity time would be less than a minute and the Request Object URI is to include a cryptographic random value of 128bit or more at the time of the writing of this specification.

(e) A third party, such as a Trust Framework Provider, provides an endpoint that provides a Request Object URI in exchange for a Request Object. The same requirements as (b) above apply. In addition, the Authorization Server MUST know out-of-band that the Client utilizes the Trust Framework Operator.

10.3. Explicit Endpoints

Although this specification does not require them, research such as [BASIN] points out that it is a good practice to explicitly state the intended interaction endpoints and the message position in the sequence in a tamper evident manner so that the intent of the initiator is unambiguous. The endpoints that come into question in this specification are :

(a) Protected Resources ("protected_resources")

(b) Authorization Endpoint ("authorization_endpoint")

(c) Redirection URI ("redirect_uri")

(d) Token Endpoint ("token_endpoint")

Further, if dynamic discovery is used, then the discovery related endpoints also come into question.

In [RFC6749], while Redirection URI is included, others are not included in the Authorization Request. As the result, the same applies to Authorization Request Object.

The lack of the link among those endpoints are sited as the cause of Cross-Phase Attacks introduced in [FETT]. An extension specification should be created as a measure to address the risk.

10.4. Risks Associated with request_uri

The introduction of "request_uri" introduces several attack possibilities.

10.4.1. DDoS Attack on the Authorization Server

A set of malicious client can launch a DoS attack to the authorization server by pointing the "request_uri" to a uri that returns extremely large content or extremely slow to respond. Under such an attack, the server may use up its resource and start failing.

Similarly, a malicious client can specify the "request_uri" value that itself points to an authorization request URI that uses "request_uri" to cause the recursive lookup.

To prevent such attack to succeed, the server should (a) check that the value of "request_uri" parameter does not point to an unexpected location, (b) check the content type of the response is "application/jwt" (c) implement a time-out for obtaining the content of "request_uri", and (d) do not perform recursive GET on the "request_uri".

10.4.2. Request URI Rewrite

The value of "request_uri" is not signed thus it can be tampered by Man-in-the-browser attacker. Several attack possibilities rise because of this, e.g., (a) attacker may create another file that the rewritten URI points to making it possible to request extra scope (b) attacker launches a DoS attack to a victim site by setting the value of "request_uri" to be that of the victim.

To prevent such attack to succeed, the server should (a) check that the value of "request_uri" parameter does not point to an unexpected location, (b) check the content type of the response is "application/jwt" (c) implement a time-out for obtaining the content of "request_uri".

11. TLS security considerations

Curent security considerations can be found in Recommendations for Secure Use of TLS and DTLS [BCP195]. This supersedes the TLS version recommendations in OAuth 2.0 [RFC6749].

12. Privacy Considerations

When the Client is being granted access to a protected resource containing personal data, both the Client and the Authorization Server need to adhere to Privacy Principles. RFC 6973 Privacy Considerations for Internet Protocols [RFC6973] gives excellent guidance on the enhancement of protocol design and implementation. The provision listed in it should be followed.

Most of the provision would apply to The OAuth 2.0 Authorization Framework [RFC6749] and The OAuth 2.0 Authorization Framework: Bearer Token Usage [RFC6750] and are not specific to this specification. In what follows, only the specific provisions to this specification are noted.

12.1. Collection limitation

When the Client is being granted access to a protected resource containing personal data, the Client SHOULD limit the collection of personal data to that which is within the bounds of applicable law and strictly necessary for the specified purpose(s).

It is often hard for the user to find out if the personal data asked for is strictly necessary. A Trust Framework Provider can help the user by examining the Client request and comparing to the proposed processing by the Client and certifying the request. After the certification, the Client, when making an Authorization Request, can submit Authorization Request to the Trust Framework Provider to obtain the Request Object URI.

Upon receiving such Request Object URI in the Authorization Request, the Authorization Server first verifies that the authority portion of the Request Object URI is a legitimate one for the Trust Framework Provider. Then, the Authorization Server issues HTTP GET request to the Request Object URI. Upon connecting, the Authorization Server MUST verify the server identity represented in the TLS certificate is

legitimate for the Request Object URI. Then, the Authorization Server can obtain the Request Object, which includes the "client_id" representing the Client.

The Consent screen MUST indicate the Client and SHOULD indicate that the request has been vetted by the Trust Framework Operator for the adherence to the Collection Limitation principle.

12.2. Disclosure Limitation

12.2.1. Request Disclosure

This specification allows extension parameters. These may include potentially sensitive information. Since URI query parameter may leak through various means but most notably through referrer and browser history, if the authorization request contains a potentially sensitive parameter, the Client SHOULD JWE [RFC7516] encrypt the request object.

Where Request Object URI method is being used, if the request object contains personally identifiable or sensitive information, the "request_uri" SHOULD be used only once, have a short validity period, and MUST have large enough entropy deemed necessary with applicable security policy unless the Request Object itself is JWE [RFC7516] Encrypted. The adequate shortness of the validity and the entropy of the Request Object URI depends on the risk calculation based on the value of the resource being protected. A general guidance for the validity time would be less than a minute and the Request Object URI is to include a cryptographic random value of 128bit or more at the time of the writing of this specification.

12.2.2. Tracking using Request Object URI

Even if the protected resource does not include a personally identifiable information, it is sometimes possible to identify the user through the Request Object URI if persistent per-user Request Object URI is used. A third party may observe it through browser history etc. and start correlating the user's activity using it. In a way, it is a data disclosure as well and should be avoided.

Therefore, per-user Request Object URI should be avoided.

13. Acknowledgements

The following people contributed to the creation of this document in the OAuth WG. (Affiliations at the time of the contribution are used.)

Sergey Beryozkin, Brian Campbell (Ping Identity), Vladimir Dzhuvinov (Connect2id), Michael B. Jones (Microsoft), Torsten Lodderstedt (YES) Jim Manico, Axel Nenker(Deutsche Telecom), Hannes Tschofenig (ARM), Ben Campbell, Kathleen Moriarty (as AD), and Steve Kent (as SECDIR).

The following people contributed to creating this document through the OpenID Connect Core 1.0 [OpenID.Core].

Brian Campbell (Ping Identity), George Fletcher (AOL), Ryo Itou (Mixi), Edmund Jay (Illumila), Michael B. Jones (Microsoft), Breno de Medeiros (Google), Hideki Nara (TACT), Justin Richer (MITRE).

In addition, the following people contributed to this and previous versions through the OAuth Working Group.

Dirk Balfanz (Google), James H. Manger (Telstra), John Panzer (Google), David Recordon (Facebook), Marius Scurtescu (Google), Luke Shepard (Facebook).

14. Revision History

Note to the RFC Editor: Please remove this section from the final RFC.

-17

- #78 Typos in content-type

-16

- Treated remaining Ben Campbell comments.

-15

- Removed further duplication

-14

- #71 Reiterate dynamic params are included.
- #70 Made clear that AS must return error.
- #69 Inconsistency of the need to sign.
- Fixed Mimetype.
- #67 Incosistence in requiring HTTPS in request uri.

- #66 Dropped ISO 29100 reference.

- #25 Removed Encrypt only option.

- #59 Same with #25.

-13

- add TLS Security Consideration section

- replace RFC7525 reference with BCP195

- moved front tag in FETT reference to fix XML structure

- changes reference from SoK to FETT

-12

- fixes #62 - Alexey Melnikov Discuss

- fixes #48 - OPSDIR Review : General - delete semicolors after list items

- fixes #58 - DP Comments for the Last Call

- fixes #57 - GENART - Remove "non-normative ... " from examples.

- fixes #45 - OPSDIR Review : Introduction - are attacks discovered or already opened

- fixes #49 - OPSDIR Review : Introduction - Inconsistent colons after initial sentence of list items.

- fixes #53 - OPSDIR Review : 6.2 JWS Signed Request Object - Clarify JOSE Header

- fixes #42 - OPSDIR Review : Introduction - readability of 'and' is confusing

- fixes #50 - OPSDIR Review : Section 4 Request Object - Clarify 'signed, encrypted, or signed and encrypted'

- fixes #39 - OPSDIR Review : Abstract - Explain/Clarify JWS and JWE

- fixed #50 - OPSDIR Review : Section 4 Request Object - Clarify 'signed, encrypted, or signed and encrypted'

- fixes #43 - OPSDIR Review : Introduction - 'properties' sounds awkward and are not exactly 'properties'

- fixes #56 - OPSDIR Review : 12 Acknowledgements - 'contribution is' => 'contribution are'

- fixes #55 - OPSDIR Review : 11.2.2 Privacy Considerations - ' It is in a way' => 'In a way, it is'

- fixes #54 - OPSDIR Review : 11 Privacy Considerations - 'and not specific' => 'and are not specific'

- fixes #51 - OPSDIR Review : Section 4 Request Object - 'It is fine' => 'It is recommended'

- fixes #47 - OPSDIR Review : Introduction - 'over- the- wire' => 'over-the-wire'

- fixes #46 - OPSDIR Review : Introduction - 'It allows' => 'The use of application security' for

- fixes #44 - OPSDIR Review : Introduction - 'has' => 'have'

- fixes #41 - OPSDIR Review : Introduction - missing 'is' before 'typically sent'

- fixes #38 - OPSDIR Review : Section 11 - Delete 'freely accessible' regarding ISO 29100

-11

- s/bing/being/

- Added history for -10

-10

- #20: KM1 -- some wording that is awkward in the TLS section.

- #21: KM2 - the additional attacks against OAuth 2.0 should also have a pointer

- #22: KM3 -- Nit: in the first line of 10.4:

- #23: KM4 -- Mention RFC6973 in Section 11 in addition to ISO 29100

- #24: SECDIR review: Section 4 -- Confusing requirements for sign+encrypt

- #25: SECDIR review: Section 6 -- authentication and integrity need not be provided if the requestor encrypts the token?

- #26: SECDIR Review: Section 10 -- why no reference for JWS algorithms?

- #27: SECDIR Review: Section 10.2 - how to do the agreement between client and server "a priori"?

- #28: SECDIR Review: Section 10.3 - Indication on "large entropy" and "short lifetime" should be indicated

- #29: SECDIR Review: Section 10.3 - Typo

- #30: SECDIR Review: Section 10.4 - typos and missing articles

- #31: SECDIR Review: Section 10.4 - Clearer statement on the lack of endpoint identifiers needed

- #32: SECDIR Review: Section 11 - ISO29100 needs to be moved to normative reference

- #33: SECDIR Review: Section 11 - Better English and Entropy language needed

- #34: Section 4: Typo

- #35: More Acknowledgment

- #36: DP - More precise qualification on Encryption needed.

-09

- Minor Editorial Nits.

- Section 10.4 added.

- Explicit reference to Security consideration (10.2) added in section 5 and section 5.2.

- , (add yourself) removed from the acknowledgment.

-08

- Applied changes proposed by Hannes on 2016-06-29 on IETF OAuth list recorded as https://bitbucket.org/Nat/oauth-jwsreq/issues/12/.

- TLS requirements added.

- Security Consideration reinforced.

- Privacy Consideration added.

- Introduction improved.

-07

- Changed the abbrev to OAuth JAR from oauth-jar.

- Clarified sig and enc methods.

- Better English.

- Removed claims from one of the example.

- Re-worded the URI construction.

- Changed the example to use request instead of request_uri.

- Clarified that Request Object parameters take precedence regardless of request or request_uri parameters were used.

- Generalized the language in 4.2.1 to convey the intent more clearly.

- Changed "Server" to "Authorization Server" as a clarification.

- Stopped talking about request_object_signing_alg.

- IANA considerations now reflect the current status.

- Added Brian Campbell to the contributors list. Made the lists alphabetic order based on the last names. Clarified that the affiliation is at the time of the contribution.

- Added "older versions of " to the reference to IE uri length limitations.

- Stopped talking about signed or unsigned JWS etc.

- 1.Introduction improved.

-06

- Added explanation on the 512 chars URL restriction.

- Updated Acknowledgements.

-05

- More alignment with OpenID Connect.

-04

- Fixed typos in examples. (request_url -> request_uri, cliend_id -> client_id)

- Aligned the error messages with the OAuth IANA registry.

- Added another rationale for having request object.

-03

- Fixed the non-normative description about the advantage of static signature.

- Changed the requirement for the parameter values in the request itself and the request object from 'MUST MATCH" to 'Req Obj takes precedence.

-02

- Now that they are RFCs, replaced JWS, JWE, etc. with RFC numbers.

-01

- Copy Edits.

15. References

15.1. Normative References

[BCP195] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, May 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[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, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC6819] Lodderstedt, T., Ed., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, DOI 10.17487/RFC6819, January 2013, <https://www.rfc-editor.org/info/rfc6819>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

15.2. Informative References

[BASIN] Basin, D., Cremers, C., and S. Meier, "Provably Repairing the ISO/IEC 9798 Standard for Entity Authentication", Journal of Computer Security - Security and Trust Principles Volume 21 Issue 6, Pages 817-846, November 2013, <https://www.cs.ox.ac.uk/people/cas.cremers/downloads/papers/BCM2012-iso9798.pdf>.

[FETT] Fett, D., Kusters, R., and G. Schmitz, "A Comprehensive Formal Security Analysis of OAuth 2.0", CCS '16 Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security Pages 1204-1215 , October 2016, <https://infsec.uni-trier.de/people/publications/paper/FettKuestersSchmitz-CCS-2016.pdf>.

[OpenID.Core] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", OpenID Foundation Standards, February 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

Authors' Addresses

Nat Sakimura
Nomura Research Institute
Otemachi Financial City Grand Cube, 1-9-2 Otemachi
Chiyoda-ku, Tokyo 100-0004
Japan

Phone: +81-3-5533-2111
Email: n-sakimura@nri.co.jp
URI: http://nat.sakimura.org/

John Bradley
Yubico
Casilla 177, Sucursal Talagante
Talagante, RM
Chile

Phone: +1.202.630.5272
Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/