Network Working Group
Internet-Draft
Intended status: Experimental
Expires: October 25, 2012

J. Richer, Ed.
The MITRE Corporation

April 23, 2012

Alternate Encoding for OAuth 2 Token Responses
draft-richer-oauth-xml-01

Abstract

This document describes a method of representing the JSON structured responses from the OAuth 2 Token Endpoint into XML and form encoded responses.

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].

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 October 25, 2012.

Copyright Notice

Table of Contents

1. Introduction

The OAuth 2 Protocol [I-D.ietf-oauth-v2] defines a standard JSON [RFC4627] encoding for structured return values from the Token Endpoint in section 5.1 of the specification when used with most flows. Additionally, the OAuth 2 specification defines a URI fragment encoding for tokens issued from the Authorization Endpoint in the Implicit Grant flow using "application/x-www-form-url-encoded" encoding in section 4.2.2.

When OAuth is being used as part of an API that is built around different encoding technologies, such as XML [W3C.CR-xml11-20021015], it is not desirable for application developers to have to parse JSON encoded objects just to handle authorization step. This extension describes a means for the client to request an alternative format for respones from the Token Endpoint and methods for the Token Endpoint to encode its responses as XML documents and form-encoded parameters. This extension makes no claim on responses from the Authorization Endpoint or other endpoints defined in OAuth2, its extensions, or profiles.

2. Content Negotiation

To request an alternate encoding from the OAuth 2 Token Endpoint, the client indicates the desired encoding through one of the following methods. Authorization Servers SHOULD support all methods but MUST support at least one so that supporting clients can be configured to request the right format. Particular formats available from a given Authorization Server MUST be documented and MAY be discoverable through some other means.

2.1. Form Parameter

In this method, the client sends the following OPTIONAL form parameter in any request to the Token Endpoint to indicate its encoding preference.

format
: OPTIONAL. The format parameter specifies the client’s desired format for responses from the token endpoint. Valid values are "json", "xml", and "form", though other extensions MAY define other valid values.

If the value of the parameter is set to "xml" and the authorization server supports XML encoding, the authorization server MUST respond to a valid token request with an HTTP 200 response, a content type of "application/xml", and HTTP body content as described in Section 3.1.

If the value of the parameter is set to "form" and the authorization server supports form encoding, the authorization server MUST respond to a valid token request with an HTTP 200 response, a content type of "application/x-www-form-encoded", and an HTTP body content as described in Section 3.2.

If the value of this parameter is "json" or the parameter is omitted entirely, the authorization server MUST respond to a valid token reqeust as defined in OAuth 2 [I-D.ietf-oauth-v2].

2.2. Accept Header

In this method, the client sends an HTTP "Accept" header to indicate to the Authorization Server what encodings it prefers as described in the HTTP specification [REF].

If the value of the header includes "application/xml" and the authorization server supports XML encoding, the authorization server MUST respond to a valid token request with an HTTP 200 response, a content type of "application/xml", and HTTP body content as described in Section 3.1.

If the value of the header includes "application/x-www-form-encoded" and the authorization server supports form encoding, the authorization server MUST respond to a valid token request with an HTTP 200 response, a content type of "application/x-www-form-url-encoded", and an HTTP body content as described in Section 3.2.

If the value of the header is "application/json" or no accept preference is otherwise given, the authorization server MUST respond to a valid token reqeust as defined in OAuth 2 [I-D.ietf-oauth-v2].

3. Encoding

All alternate forms of encoding MUST account for all elements of a token as specified in OAuth2.

3.1. XML

For a full description of the transformation rules, see Appendix A (Appendix A).

The response MUST use a single XML root element with a node name of "oauth" to represent the anonymous root JSON object specified in the OAuth JSON response.

The response SHOULD NOT include a default namespace.

All elements of the JSON object MUST be encoded as XML elements, with values encoded as CDATA within each element.

3.2. Form Encoding

For a full description of the transformation rules, see Appendix B (Appendix B).

The form encoding MUST follow the same encoding rules as defined in Section 4.2.2 of OAuth2.

All values of the JSON response MUST be encoded as key-value pairs.

4. Examples

Below are examples of encoding different OAuth JSON objects with XML. All line breaks are for display purposes only.

4.1. Standard OAuth Token

A standard OAuth JSON-encoded token response (example from OAuth2 Core):

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "example_parameter":"example_value"
}
```

Can be encoded in as the following XML response document:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: no-store

<oauth>
  <access_token>2YotnFZFEjr1zCsicMWpAA</access_token>
  <token_type>example</token_type>
  <expires_in>3600</expires_in>
  <refresh_token>tGzv3JOkF0XG5Qx2TlKWIA</refresh_token>
  <example_parameter>example_value</example_parameter>
</oauth>
```

The same response can be encoded in form encoding a follows:

```
HTTP/1.2 200 OK
Content-Type: application/x-www-form-encoded
Cache-Control: no-store

access_token=2YotnFZFEjr1zCsicMWpAA&token_type=example&
expires_in=3600&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA&
example_parameter=example_value
```

5. IANA Considerations

This document makes no request of IANA.

6. Security Considerations

There are no additional security considerations.

7. Acknowledgements

Thanks to Eve Maler, Joseph Holsten, Tim Brody, and the OAuth Working Group for feedback.

8. Normative References

[I-D.ietf-oauth-v2]

Hammer-Lahav, E., Recordon, D., and D. Hardt, "The OAuth 2.0 Authorization Protocol", draft-ietf-oauth-v2-23 (work in progress), January 2012.

[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.

[W3C.CR-xml11-20021015] Cowan, J., "Extensible Markup Language (XML) 1.1", W3C CR CR-xml11-20021015, October 2002.

Appendix A. General XML Encoding Rules

This Appendix defines encodings for different parts of the JSON data model in XML equivalents to facilitate structured extensions to the OAuth2 JSON token response. Since this JSON response MAY include elements such as JSON objects or arrays, a server wishing to support such extended responses and XML encoding MUST use these encoding rules to translate them.

A.1. Objects and Members

JSON objects SHALL be encoded by using XML Elements. The object itself SHALL be represented by the root elment of an XML subtree. All members of the object SHALL be represented by sub-elements of the root element. The key of the member pair SHALL be the node name of the XML Element, and the value of the member pair SHALL be encoded as the content of the XML Element.

A.2. Type Identifiers

All elements MAY have an OPTIONAL "type" attribute, which has a valid value of "object", "string", "number", or "array". These attributes can be used to differentiate between otherwise potentially ambiguous encodings (Appendix A.6), though the most common cases will not need them.

A.3. Strings and Numbers

Strings and numbers SHALL be encoded as CDATA within their enclosing element. These values MUST be properly escaped XML CDATA, and MAY be represented using <[CDATA[...]]> encoding.

A.4. Arrays

Arrays SHALL be represented using repeated, sibling XML Element nodes (nodes with the same node name). The order of the array is encoded using document order of the array elements.

A.5. Namespace

This extension does not define a required namespace for the OAuth XML encoding, and a supporting server SHOULD not use a namespace.

A.6. Information Loss

This encoding scheme is intended to give a clear an intuitive mapping between JSON and XML data structures. However, the mapping between the two formats is not exact and some information loss may occur, and round-trip translation between the two formats MUST NOT be depended upon.

1. Both strings and numbers (Appendix A.3) in JSON are represented as CDATA in XML. Without type identifiers (Appendix A.2) there is no clear way to differentiate between the two in the XML encoding.

2. Arrays (Appendix A.4) in JSON are represented by repeated elements in XML. There is therefore no reliable way to distinguish between a single-element array and a standalone string or number value in the XML encoding, as both would be encoded the same way.

A.7. Examples

Line breaks are for display purposes only.

The example above, with type attributes (Appendix A.2) in place:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: no-store

<oauth type="object">
  <access_token type="string">2YotnFZFEjr1zCsicMWpAA</access_token>
  <token_type type="string">example</token_type>
  <expires_in type="number">3600</expires_in>
  <refresh_token type="string">tGzv3JOkF0XG5Qx2TlKWIA</refresh_token>
  <example_parameter type="string">example_value</example_paramter>
</oauth>
```

This example uses both objects and arrays to support a complicated, fictional example extension to the OAuth protocol:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"example",
  "expires_in":3600,
  "refresh_token":"tGzv3JOkF0XG5Qx2TlKWIA",
  "ext_value": "extension",
  "ext_list": [ 1, 2, "three" ],
  "ext_object": {
        "member1": "value1",
        "memberlist": [ "A", "B", "C"],
        "member3": 3,
        "memberobj": {
            "a": "first",
            "b": "second",
            "c": "third"
        }
  }
}
```

The above is encoded into XML as follows (without using type attributes):

```
HTTP/1.1 200 OK
Content-Type: application/xml
Cache-Control: no-store

<oauth>
  <access_token>2YotnFZFEjr1zCsicMWpAA</access_token>
  <token_type>example</token_type>
  <expires_in>3600</expires_in>
  <refresh_token>tGzv3JOkF0XG5Qx2TlKWIA</refresh_token>
  <ext_value>extension</ext_value>
  <ext_list>1</ext_list>
  <ext_list>2</ext_list>
  <ext_list>three</ext_list>
  <ext_object>
        <member1>value1</member>
        <memberlist>A</memberlist>
        <memberlist>B</memberlist>
        <memberlist>C</memberlist>
        <member3>3</member3>
        <memberobj>
            <a>first</a>
            <b>second</b>
            <c>third</c>
        </memberobj>
  </ext_object>
</oauth>
```

Appendix B. General Form Encoding Rules

This Appendix defines encodings for different parts of the JSON data model in form encoded equivalents to facilitate structured extensions to the OAuth2 JSON token response. Since this JSON response MAY include elements such as JSON objects or arrays, a server wishing to support such extended responses and form encoding MUST use these encoding rules to translate them. These encoding rules MAY be used to extend the response of the Authorization Endpoint in the Implicit flow.

B.1. Objects and Members

JSON objects SHALL be represented by encoding all members as separate form parameters. Sub-objects SHALL be encoded by a dot-notation

syntax, with the member name of a sub-object being appended to the name of its containing object member, separated by a single period.

B.2. Strings and Numbers

All String and Number values SHALL be encoded as simple string values.

B.3. Arrays

Arrays SHALL be encoded by repeating the member name for each value in the array. The order of the array is encoded by the presentation order of the values in the response.

B.4. Information Loss

This encoding scheme is intended to give a clear an intuitive mapping between JSON and form encoded data structures. However, the mapping between the two formats is not exact and some information loss may occur, and round-trip translation between the two formats MUST NOT be depended upon.

1. Both strings and numbers (Appendix B.2) in JSON are represented as strings in the form encoding, and there is no clear way to differentiate between the two in the form encoding.

2. Arrays (Appendix B.3) in JSON are represented by repeated elements in the form encoding. There is therefore no reliable way to distinguish between a single-element array and a standalone string or number value in the form encoding, as both would be encoded the same way.

B.5. Examples

This example encodes the fictionally extended OAuth token response above. Line breaks are for display purposes only.

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-encoded
Cache-Control: no-store

access_token=2YotnFZFEjr1zCsicMWpAA&token_type=example&
expires_in=3600&refresh_token=tGzv3JOkF0XG5Qx2TlKWIA&
ext_value=extension&ext_list=1&ext_list=2&ext_list=three&
ext_object.member1=value1&ext_object.memberlist=A&
ext_object.memberlist=B&ext_object.memberlist=C&
ext_object.member3=3&ext_object.memberobj.a=first&
ext_object.memberobj.b=second&ext_object.memberobj.c=third
```

Author's Address

Justin Richer (editor)
The MITRE Corporation

Email: jricher@mitre.org