Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 9, 2015

T. Hardjono, Ed.
MIT
E. Maler
ForgeRock
M. Machulak
Cloud Identity
D. Catalano
Oracle

January 5, 2015

OAuth 2.0 Resource Set Registration
draft-hardjono-oauth-resource-reg-04

Abstract

This specification defines a resource set registration mechanism between an OAuth 2.0 authorization server and resource server. The resource server registers information about the semantics and discovery properties of its resources with the authorization server. This revision of the specification is part of the UMA "candidate V1.0" process.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

There are various circumstances under which an OAuth 2.0 [OAuth2] resource server may need to communicate information about its protected resources to its authorization server:

o In some OAuth 2.0 deployments, the resource server and authorization server are operated by the same organization and deployed in the same domain, but many resource servers share a single authorization server (a security token service (STS) component). Thus, even though the trust between these two is typically tightly bound, there is value in defining a singular

standardized resource protection communications interface between the authorization server and each of the resource servers.

- In some deployments of OpenID Connect [OpenIDConnect], which has a dependency on OAuth 2.0, the OpenID Provider (OP) component is a specialized version of an OAuth authorization server that brokers availability of user attributes by dealing with an ecosystem of attribute providers (APs). These APs effectively function as third-party resource servers. Thus, there is value in defining a mechanism by which all of the third-party APs can communicate with a central OP, as well as ensuring that trust between the authorization server and resource servers is able to be established in a dynamic, loosely coupled fashion.

- In some deployments of User-Managed Access [UMA], which has a dependency on OAuth 2.0, an end-user resource owner (the "user" in UMA) may choose their own authorization server as an independent cloud-based service, along with using any number of resource servers that make up their "personal cloud". Thus, there is value in defining a mechanism by which all of the third-party resource servers can outsource resource protection (and potentially discovery) to a central authorization server, as well as ensuring that trust between the authorization server and resource servers is able to be established by the resource owner in a dynamic, loosely coupled fashion.

This specification defines an API through which the resource server can register information about resource sets with the authorization server.

1.1. Notational Conventions

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].

Unless otherwise noted, all protocol properties and values are case sensitive. JSON [JSON] data structures defined by this specification MAY contain extension properties that are not defined in this specification. Any entity receiving or retrieving a JSON data structure SHOULD ignore extension properties it is unable to understand. Extension names that are unprotected from collisions are outside the scope of this specification.

1.2. Terminology

This specification introduces the following new terms and enhancements of OAuth term definitions.

resource set One or more resources that the resource server manages as a set, abstractly. A resource set may be a single API endpoint, a set of API endpoints, a classic web resource such as an HTML page, and so on. Defining this concept enables registering data about it, including, most importantly, scopes but also other data.

scope A bounded extent of access that is possible to perform on a resource set. In authorization policy terminology, a scope is one of the potentially many "verbs" that can logically apply to a resource set ("object"). This specification enhances the OAuth concept of a "scope" by defining scopes as applying to particular registered resource sets, rather than leaving the relevant resources (such as API endpoints or URIs) implicit. A resource set can have any number of scopes, which together describe the universe of actions that _can be_ taken on this protected resource set. For example, a resource set representing a status update API might have scopes that include adding an update or reading updates. A resource set representing a photo album might have scopes that include viewing a slideshow or printing the album. The resource server registers resource sets and their scopes when there is not yet any particular client in the picture.

resource set registration endpoint The endpoint defined by this specification at which the resource server registers resource sets it wants the authorization server to know about. The operations available at this endpoint constitute a resource set registration API (see Section 2.3).

1.3. Authorization Server Configuration Data

If the authorization server declares its endpoints and any other configuration data in a machine-readable form, it SHOULD convey its resource set registration endpoint in this fashion as well.

2. Resource Set Registration

This specification defines a resource set registration API. The endpoint for this API SHOULD also require some form of authentication to access this endpoint, such as Client Authentication as described in [OAuth2] or a separate OAuth access token. The methods of

managing and validating these authentication credentials are out of scope of this specification.

For any of the resource owner's sets of resources this authorization server needs to be aware of, the resource server MUST register these resource sets at the authorization server's registration endpoint.

2.1. Scope Descriptions

A scope description is a JSON document with the following properties:

name REQUIRED. A human-readable string describing some scope (extent) of access. This name MAY be used by the authorization server in any user interface it presents to the resource owner.

icon_uri OPTIONAL. A URI for a graphic icon representing the scope. The referenced icon MAY be used by the authorization server in any user interface it presents to the resource owner.

For example, this scope description characterizes a scope that involves reading or viewing resources (vs. creating them or editing them in some fashion):

```
{
  "name" : "View",
  "icon_uri" : "http://www.example.com/icons/reading-glasses"
}
```

See Section 8 for a long-form example of scope descriptions used in resource set registration.

2.2. Resource Set Descriptions

The resource server defines a resource set that the authorization server needs to be aware of by registering a resource set description at the authorization server. This registration process results in a unique identifier for the resource set that the resource server can later use for managing its description.

The resource server is free to use its own methods of describing resource sets. A resource set description is a JSON document with the following properties:

name REQUIRED. A human-readable string describing a set of one or more resources. This name MAY be used by the authorization server in its resource owner user interface for the resource owner.

uri OPTIONAL. A URI that provides the network location for the resource set being registered. For example, if the resource set corresponds to a digital photo, the value of this property could be an HTTP-based URI identifying the location of the photo on the web. The authorization server can use this information in various ways to inform clients about a resource set's location.

type OPTIONAL. A string uniquely identifying the semantics of the resource set. For example, if the resource set consists of a single resource that is an identity claim that leverages standardized claim semantics for "verified email address", the value of this property could be an identifying URI for this claim.

scopes REQUIRED. An array of strings, any of which MAY be a URI, indicating the available scopes for this resource set. URIs MUST resolve to scope descriptions as defined in Section 2.1. Published scope descriptions MAY reside anywhere on the web; a resource server is not required to self-host scope descriptions and may wish to point to standardized scope descriptions residing elsewhere. It is the resource server's responsibility to ensure that scope description documents are accessible to authorization servers through GET calls to support any user interface requirements. The resource server and authorization server are presumed to have separately negotiated any required interpretation of scope handling not conveyed through scope descriptions.

icon_uri OPTIONAL. A URI for a graphic icon representing the resource set. The referenced icon MAY be used by the authorization server in its resource owner user interface for the resource owner.

For example, this description characterizes a resource set (a photo album) that can potentially be only viewed, or alternatively to which full access can be granted; the URIs point to scope descriptions as defined in Section 2.1:

```
{
  "name" : "Photo Album",
  "icon_uri" : "http://www.example.com/icons/flower.png",
  "scopes" : [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/all"
  ],
  "type" : "http://www.example.com/rsets/photoalbum"
}
```

2.3. Resource Set Registration API

The resource server uses the RESTful API at the authorization server's resource set registration endpoint to create, read, update, and delete resource set descriptions, along with retrieving lists of such descriptions.

(Note carefully the similar but distinct senses in which the word "resource" is used in this section. The resource set descriptions are themselves managed as web resources at the authorization server through this API.)

The authorization server MUST present an API for registering resource set descriptions at a set of URIs with the following structure:

```
{rsreguri}/resource_set/{rsid}
```

The method of authentication that the resource server uses SHOULD sufficient context to distinguish between identical resource set identifiers assigned by different resource servers.

The components of these URIs are defined as follows:

{rsreguri} The authorization server's resource set registration endpoint as advertised in its configuration data (see Section 1.3).

{rsid} An identifier for a resource set description. It is RECOMMENDED to obscure resource set identifiers in order to avoid leaking personally identifiable information to clients in case a client is exposed to resource set or scope description information.

Following is a summary of the five registration operations the authorization server is REQUIRED to support. Each is defined in its own section below. All other methods are unsupported. This API uses ETag and If-Match to ensure the desired resource at the authorization server is targeted.

- Create resource set description: PUT /resource_set/{rsid}

- Read resource set description: GET /resource_set/{rsid}

- Update resource set description: PUT /resource_set/{rsid} with If-Match

- Delete resource set description: DELETE /resource_set/{rsid}

o List resource set descriptions: GET /resource_set/ with If-Match

If the request to the resource set registration endpoint is incorrect, then the authorization server responds with an error message by including one of the following error codes with the response (see Section 3):

unsupported_method_type The resource server request used an unsupported HTTP method. The authorization server MUST respond with the HTTP 405 (Method Not Allowed) status code and MUST fail to act on the request.

not_found The resource set requested from the authorization server cannot be found. The authorization server MUST respond with HTTP 404 (Not Found) status code.

precondition_failed The resource set that was requested to be deleted or updated at the authorization server did not match the If-Match value present in the request. The authorization server MUST respond with HTTP 412 (Precondition Failed) status code and MUST fail to act on the request.

2.3.1. Create Resource Set Description

Adds a new resource set description using the PUT method. If the request is successful, the authorization server MUST respond with a status message that includes an ETag header and an _id property for managing resource set description versioning.

Form of a "create resource set description" HTTP request, with an access token in the header:

```
PUT /resource_set/{rsid} HTTP/1.1
Content-Type: application/json
Authorization: Bearer 204c69636b6c69
...

(body contains JSON resource set description to be created)
```

Form of a successful HTTP response:

```
HTTP/1.1 201 Created
Content-Type: application/json
ETag: ...
...

{
  "_id" : (id of created resource set)
}
```

On successful registration, the authorization server MAY return a redirect policy URI to the resource server in a property with the name "policy_uri". This URI allows the resource server to redirect the resource owner to a specific user interface within the authorization server where the resource owner can immediately set or modify access policies for the resource set that was just registered.

Form of a successful HTTP response:

```
HTTP/1.1 201 Created
Content-Type: application/json
ETag: ...
...

{
  "_id" : (id of created resource set),
  "policy_uri" : "http://as.example.com/rs/222/resource/333/policy"
}
```

2.3.2. Read Resource Set Description

Reads a previously registered resource set description using the GET method. If the request is successful, the authorization server MUST respond with a status message that includes an ETag header and an _id property for managing resource set description versioning.

Form of a "read resource set description" HTTP request, with an access token in the header:

```
GET /resource_set/{rsid} HTTP/1.1
Authorization: Bearer 204c69636b6c69
...
```

Form of a successful HTTP response:

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

(body contains JSON resource set description, including _id)

If the referenced resource does not exist, the authorization server MUST produce an error response with an error property value of "not_found", as defined in Section 2.3.

On successful read, the authorization server MAY return a redirect policy URI to the resource server in a property with the name "policy_uri". This URI allows the resource server to redirect the resource owner to a specific user interface within the authorization server where the resource owner can immediately set or modify access policies for the resource set that was read.

2.3.3. Update Resource Set Description

Updates a previously registered resource set description using the PUT method. If the request is successful, the authorization server MUST respond with a status message that includes an ETag header and _id and _rev properties for managing resource set description versioning.

Form of an "update resource set description" HTTP request, with an access token in the header:

```
PUT /resource_set/{rsid} HTTP/1.1
Content-Type: application/json
If-Match: (entity tag of resource)
Authorization: Bearer 204c69636b6c69
...
```

(body contains JSON resource set description to be updated)

Form of a successful HTTP response:

```
HTTP/1.1 204 No Content
ETag: "2"
...
```

If the entity tag does not match, the authorization server MUST produce an error response with an error property value of "precondition_failed", as defined in Section 2.3.

On successful update, the authorization server MAY return a redirect policy URI to the resource server in a property with the name "policy_uri". This URI allows the resource server to redirect the user to a specific user interface within the authorization server where the user can immediately set or modify access policies for the resource set that was just updated.

2.3.4. Delete Resource Set Description

Deletes a previously registered resource set description using the DELETE method, thereby removing it from the authorization server's protection regime.

Form of a "delete resource set description" HTTP request, with an access token in the header:

```
DELETE /resource_set/{rsid}
If-Match: (entity tag of resource)
Authorization: Bearer 204c69636b6c69
...
```

Form of a successful HTTP response:

```
HTTP/1.1 204 No content
...
```

As defined in Section 2.3, if the referenced resource does not exist the authorization server MUST produce an error response with an error property value of "not_found", and if the entity tag does not match the authorization server MUST produce an error response with an error property value of "precondition_failed".

2.3.5. List Resource Set Descriptions

Lists all previously registered resource set identifiers for this user using the GET method. The authorization server MUST return the list in the form of a JSON array of {rsid} values.

The resource server uses this method as a first step in checking whether its understanding of protected resources is in full synchronization with the authorization server's understanding.

Form of a "list resource set descriptions" HTTP request, with an access token in the header:

```
GET /resource_set HTTP/1.1
Authorization: Bearer 204c69636b6c69
...
```

HTTP response:

```
HTTP/1.1 200 OK
...
```

(body contains JSON array of {rsid} values)

3. Error Messages

When a resource server attempts to access the resource set registration endpoint at the authorization server, if the request is successfully authenticated by OAuth means, but is invalid for another reason, the authorization server produces an error response by adding the following properties to the entity body of the HTTP response:

error REQUIRED. A single error code, as noted in the API definition. Value for this property is defined in the specific authorization server endpoint description.

error_description OPTIONAL. A human-readable text providing additional information, used to assist in the understanding and resolution of the error occurred.

error_uri OPTIONAL. A URI identifying a human-readable web page with information about the error, used to provide the end-user with additional information about the error.

4. Security Considerations

This specification relies on OAuth for API security and shares its security and vulnerability considerations.

5. Privacy Considerations

The communication between the authorization server and resource server may expose personally identifiable information. The context in which this API is used SHOULD deal with its own unique privacy considerations.

6. Conformance

This specification makes optional normative reference to [OAuth2] for API protection. This specification is anticipated to be used as a module in higher-order specifications, where additional constraints and profiling may appear.

7. IANA Considerations

This document makes no request of IANA.

8. Example of Registering Resource Sets

The following example illustrates the intent and usage of resource set descriptions and scope descriptions as part of resource set registration in the context of [UMA].

This example contains some steps that are exclusively in the realm of user experience rather than web protocol, to achieve realistic illustration. These steps are labeled "user experience only". Some other steps are exclusively internal to the operation of the entity being discussed. These are labeled "internal only".

A resource owner, Alice Adams, has just uploaded a photo of her new puppy to a resource server, Photoz.example.com, and wants to ensure that this specific photo is not publicly accessible.

Alice has already introduced this resource server to her authorization server, CopMonkey.example.com. However, Alice has not previously instructed Photoz to use CopMonkey to protect any photos of hers.

Alice has previously visited CopMonkey to map a default "do not share with anyone" policy to any resource sets registered by Photoz, until such time as she maps some other more permissive policies to those resources. (User experience only. This may have been done at the time Alice introduced the resource server to the authorization server, and/or it could have been a global or resource server-specific preference setting. A different constraint or no constraint at all might be associated with newly protected resources.) Other kinds of policies she may eventually map to particular photos or albums might be "Share only with husband@email.example.net" or "Share only with people in my 'family' group".

Photoz itself has a publicly documented application-specific API that offers two dozen different methods that apply to single photos, such as "addTags" and "getSizes", but rolls them up into two photo-related scopes of access: "view" (consisting of various read-only operations) and "all" (consisting of various reading, editing, and printing operations). It defines two scope descriptions that represent these scopes, which it is able to reuse for all of its users (not just Alice), and ensures that these scope description documents are available through HTTP GET requests that may be made by authorization servers.

The "name" property values are intended to be seen by Alice when she maps authorization constraints to specific resource sets and actions while visiting CopMonkey, such that Alice would see the strings "View Photo and Related Info" and "All Actions", likely accompanied by the referenced icons, in the CopMonkey interface. (Other users of Photoz might similarly see the same labels at CopMonkey or whatever other authorization server they use. Photoz could distinguish natural-language labels per user if it wishes, by pointing to scopes with differently translated names.)

Example of the viewing-related scope description document available at http://photoz.example.com/dev/scopes/view:

```
{
  "name" : "View Photo and Related Info",
  "icon_uri" : "http://www.example.com/icons/reading-glasses.png"
}
```

Example of the broader scope description document available at http://photoz.example.com/dev/scopes/all:

```
{
  "name" : "All Actions",
  "icon_uri" : "http://www.example.com/icons/galaxy.png"
}
```

While visiting Photoz, Alice selects a link or button that instructs the site to "Protect" or "Share" this single photo (user experience only; Photoz could have made this a default or preference setting).

As a result, Photoz defines for itself a resource set that represents this photo (internal only; Photoz is the only application that knows how to map a particular photo to a particular resource set). Photoz also prepares the following resource set description, which is specific to Alice and her photo. The "name" property value is intended to be seen by Alice in mapping authorization policies to specific resource sets and actions when she visits CopMonkey. Alice would see the string "Steve the puppy!", likely accompanied by the referenced icon, in the CopMonkey interface. The possible scopes of access on this resource set are indicated with URI references to the scope descriptions, as shown just above.

```
{
  "name" : "Steve the puppy!",
  "icon_uri" : "http://www.example.com/icons/flower",
  "scopes" : [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/all"
  ]
}
```

Photoz uses the "create resource set description" method of CopMonkey's standard OAuth resource set registration API, presenting its Alice-specific access token to use the API to register and assign an identifier to the resource set description.

```
PUT /resource_set/112210f47de98100 HTTP/1.1
Content-Type: application/json
...

{
  "name" : "Steve the puppy!",
  "icon_uri" : "http://www.example.com/icons/flower.png",
  "scopes" : [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/all"
  ]
}
```

If the registration attempt succeeds, CopMonkey responds in the following fashion.

```
HTTP/1.1 201 Created
Content-Type: application/json
ETag: "1"
...

{
  "_id" : "112210f47de98100"
}
```

At the time Alice indicates she would like this photo protected, Photoz can choose to redirect Alice to CopMonkey for further policy setting, access auditing, and other authorization server-related tasks (user experience only).

Once it has successfully registered this description, Photoz is responsible for outsourcing protection to CopMonkey for access attempts made to this photo.

Over time, as Alice uploads other photos and creates and organizes photo albums, Photoz can use additional methods of the resource set registration API to ensure that CopMonkey's understanding of Alice's protected resources matches its own.

For example, if Photoz suspects that somehow its understanding of the resource set has gotten out of sync with CopMonkey's, it can ask to read the resource set description as follows.

```
GET /resource_set/112210f47de98100 HTTP/1.1
Host: as.example.com
...
```

CopMonkey responds with the full content of the resource set description, including its _id, as follows:

Example of an HTTP response to a "read resource set description" request, containing a resource set description from the authorization server:

```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: "1"
...

{
  "_id" : "112210f47de98100",
  "name" : "Photo album",
  "icon_uri" : "http://www.example.com/icons/flower.png",
  "scopes" : [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/all"
  ]
}
```

If for some reason Photoz and CopMonkey have gotten dramatically out of sync, Photoz can ask for the list of resource set identifiers CopMonkey currently knows about:

```
GET /resource_set HTTP/1.1
Host: as.example.com
...
```

CopMonkey's response might look as follows:

```
HTTP/1.1 200 OK
...

[ "112210f47de98100", "34234df47eL95300" ]
```

If Alice later changes the photo's title (user experience only) on Photoz from "Steve the puppy!" to "Steve on October 14, 2011", Photoz would use the "update resource set description" method to ensure that Alice's experience of policy-setting at CopMonkey remains consistent with what she sees at Photoz. Following is an example of this request.

```
PUT /resource_set/112210f47de98100 HTTP/1.1
Content-Type: application/json
Host: as.example.com
If-Match: "1"
...

{
  "name" : "Steve on October 14, 2011",
  "icon_uri" : "http://www.example.com/icons/flower.png",
  "scopes" : [
    "http://photoz.example.com/dev/scopes/view",
    "http://photoz.example.com/dev/scopes/all"
  ]
}
```

CopMonkey would respond as follows.

```
HTTP/1.1 201 Created
Content-Type: application/json
ETag: "2"
...

{
  "_id" : "112210f47de98100"
}
```

There are other reasons Photoz might want to update resource set descriptions, having nothing to do with Alice's actions or wishes. For example, it might extend its API to include new features, and want to add new scopes to all of Alice's and other users' resource set descriptions.

if Alice later decides to entirely remove sharing protection (user experience only) on this photo while visiting Photoz, ensuring that the public can get access without any protection, Photoz is

responsible for deleting the relevant resource set registration, as follows:

```
DELETE /resource_set/112210f47de98100 HTTP/1.1
Host: as.example.com
If-Match: "2"
...
```

9. Acknowledgments

The following people made significant text contributions to the specification:

- o Paul C. Bryan, ForgeRock US, Inc. (former editor)
- o Mark Dobrinic, Cozmanova
- o George Fletcher, AOL
- o Lukasz Moren, Cloud Identity Ltd
- o Christian Scholz, COMlounge GmbH (former editor)
- o Mike Schwartz, Gluu
- o Jacek Szpot, Newcastle University

Additional contributors to this specification include the Kantara UMA Work Group participants, a list of whom can be found at [UMAnitarians].

10. References

10.1. Normative References

[JSON]   Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", March 2014, <https://tools.ietf.org/html/rfc7159>.

[OAuth2]   Hardt, D., "The OAuth 2.0 Authorization Framework", October 2012, <http://tools.ietf.org/html/rfc6749>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10.2. Informative References

[OpenIDConnect]
Sakimura, N., "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <http://openid.net/specs/openid-connect-core-1_0.html>.

[UMA] Hardjono, T., "User-Managed Access (UMA) Profile of OAuth 2.0", December 2014, <http://docs.kantarainitiative.org/uma/draft-uma-core.html>.

[UMAnitarians]
Maler, E., "UMA Participant Roster", December 2014, <http://kantarainitiative.org/confluence/display/uma/Participant+Roster>.

10.3. URIs

[1] http://kantarainitiative.org/confluence/display/uma/UMA+1.0+Core+Protocol

Appendix A. Document History

NOTE: To be removed by RFC editor before publication as an RFC.

See [1] for a list of code-breaking and other major changes made to this specification at various revision points.

Authors' Addresses

Thomas Hardjono (editor)
MIT

Email: hardjono@mit.edu

Eve Maler
ForgeRock

Email: eve.maler@forgerock.com

Maciej Machulak
Cloud Identity

Email: maciej.machulak@cloudidentity.co.uk

Domenico Catalano
Oracle

Email: domenico.catalano@oracle.com