Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2012

T. Hardjono, Ed.
MIT

October 16, 2011

User-Managed Access (UMA) Core Protocol
draft-hardjono-oauth-umacore-01

Abstract

This specification defines the User-Managed Access (UMA) core protocol. This protocol provides a method for users to control access to their protected resources, residing on any number of host sites, through an authorization manager that governs access decisions based on user policy.

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 18, 2012.

Copyright Notice

Table of Contents

1. Introduction

The User-Managed Access (UMA) core protocol provides a method based on [OAuth2] (currently draft 16) for users to control access to their protected resources, residing on any number of host sites, through a single authorization manager (AM) that governs access decisions based on user policy.

There are numerous use cases for UMA, where a resource owner nominates a third party to control access to these resources potentially without the real-time presence of the resource owner. A typical example is the following. A web user (authorizing user) can authorize a web app (requester) to gain one-time or ongoing access to a resource containing his home address stored at a "personal data store" service (host), by telling the host to act on access decisions made by his authorization decision-making service (authorization manager or AM). The requesting party might be an e-commerce company whose site is acting on behalf of the user himself to assist him/her in arranging for shipping a purchased item, or it might be his friend who is using an online address book service to collect addresses, or it might be a survey company that uses an online service to compile population demographics. Other scenarios and use cases for UMA usage can be found in [UMA-usecases] and [UMA-userstories].

In enterprise settings, application access management often involves letting back-office applications serve only as policy enforcement points (PEPs), depending entirely on access decisions coming from a central policy decision point (PDP) to govern the access they give to requesters. This separation eases auditing and allows policy administration to scale in several dimensions. UMA makes use of a separation similar to this, letting the authorizing user serve as a policy administrator crafting authorization strategies on his or her own behalf.

The UMA protocol profiles, extends, and embeds [OAuth2] in various ways. An AM can be thought of as an enhanced OAuth authorization server; a host as an enhanced resource server; and a requester as an enhanced client, acquiring an access token and the requisite authorization to access a protected resource at the host.

The UMA protocol has three broad phases, as shown in Figure 1.

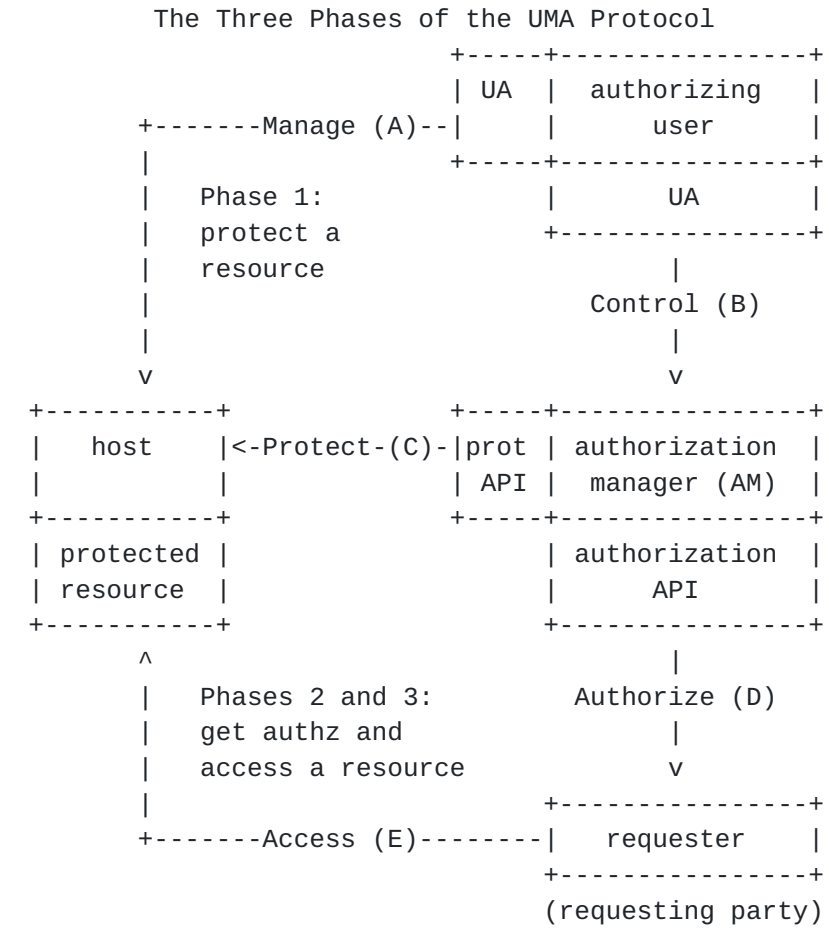

Figure 1

In broad strokes, the phases are as follows:

1. Protect a resource (described in Section 2).

2. Get authorization (described in Section 3).

3. Access a resource (described along with Phase 2 in Section 3).

In more detail, the phases work as follows:

1. _Protect a resource:_ The authorizing user has chosen to use a host for managing online resources ("A"), and introduces this host to an AM using an OAuth-mediated interaction that results in the AM giving the host an access token. The host uses AM's protection API to tell the AM what sets of resources to protect ("C"). Out of band of the UMA protocol, the authorizing user instructs the AM what policies to attach to the registered resource sets ("B"). Requesters are not yet in the picture.

2. _Get authorization:_ This phase involves the requester, host, and AM. It may also involve synchronous action by the authorizing user if this person is the same person as the requesting party. This phase is dominated by a loop of activity in which the requester approaches the host seeking access to a protected resource ("E"), is sent to obtain an access token from the AM if it does not have one, and then must demonstrate to the AM that it satisfies the user's authorization policy governing the sought-for resource and scope of access if it does not already have the required access permission ("D").

3. _Access a resource:_ This phase involves the requester successfully presenting an access token that has sufficient permission associated with it to the host in order to gain access to the desired resource ("E"). In this sense, it is the "happy path" within phase 2.

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 the protocol parameter names and values are case sensitive.

The assignment in this document of URI labels is temporary, awaiting final standardization in the eventual standards body within which this specification is taken up as a work item.

1.2. Basic Terminology

UMA introduces the following terms, utilizing OAuth and other identity and access management concepts.

authorizing user
: An UMA-defined variant of an OAuth end-user resource owner; a web user who configures an authorization manager with policies that control how it assigns access permissions to requesters for a protected resource.

authorization manager (AM)
: An UMA-defined variant of an OAuth authorization server that carries out an authorizing user's policies governing access to a protected resource.

protected resource
: An access-restricted resource at a host, which is being policy-protected by an AM.

host
: An UMA-defined variant of an OAuth resource server that enforces access to the protected resources it hosts, as governed by an authorization manager.

host access token
: An access token representing the authorizing user's consent for a host to trust a particular authorization manager for control over authorizations to access protected resources hosted there.

claim
: A statement of the value or values of one or more identity attributes of a requesting party. A requesting party may need to provide claims to an authorization manager in order to satisfy policy and gain permission for access to a protected resource.

requester
: An UMA-defined variant of an OAuth client that seeks access to a protected resource.

requester access token
: An access token that can be associated with permissions to access particular resources at a host on behalf of a particular requesting party.

requesting party
: A web user, or a corporation or other legal person, that uses a requester to seek access to a protected resource. If the requesting party is a natural person, it may or may not be the same person as the authorizing user.

resource set description
: A JSON-formatted document that represents a set of one or more resources to be AM-protected and maps available scopes to them. The host registers a resource set by providing this document to the AM.

scope description A JSON-formatted document that represents a
: bounded scope (extent) of access on a particular resource set. The host refers to this type of document from within its registered resource set descriptions and permissions.

token status description A JSON-formatted document that represents the currently valid permissions for access associated with a requester access token.

permission A scope of access over a particular resource set at a particular host that is being asked for by a requester (a requested permission) or that has been granted by an AM (a granted permission).

1.3. Endpoints, Endpoint Protection, and Tokens

As indicated in Section 1, various UMA entities present APIs for other UMA entities to use. These APIs are as follows:

- The AM presents a _protection API_ to the host, as standardized by this specification. This API is OAuth-protected, requiring a host access token (issued by the AM) for successful access (see Section 2.3 for this issuance process).

- The AM presents an _authorization API_ to the requester, as standardized by this specification. This API is OAuth-protected, requiring a requester access token (issued by the AM) for successful access (see Section 3.2 for this issuance process).

- The host presents a _protected resource_ to the requester, which can be considered -- and may in fact be -- an application-specific or proprietary API. This API is UMA-protected, requiring a requester access token (issued by the AM) and sufficient permissions (also issued by the AM) for successful access (see Section 3.5 for this latter issuance process).

The AM presents the following endpoints to the host as part of its protection API:

host access token endpoint Part of standard OAuth, as profiled by UMA. The endpoint at which the host asks for a host access token on the authorizing user's behalf. (The AM may also choose to issue a refresh token.) It will use this token to gain access to the other protection API endpoints.

host user authorization endpoint Part of standard OAuth, as profiled by UMA. The endpoint to which the host redirects the authorizing user to authorize the host to use this AM for protecting resources, if the OAuth authorization code grant type is being used.

resource set registration endpoint The endpoint at which the host registers resource sets it wants the AM to protect.

permission registration endpoint The endpoint at which the host registers permissions that it anticipates a requester will shortly be asking for from the AM.

token status endpoint The endpoint at which the host submits requester access tokens that have accompanied an access request, to learn what currently valid permissions are associated with them.

The AM presents the following endpoints to the requester as part of its authorization API:

requester access token endpoint Part of standard OAuth, as profiled by UMA. The endpoint at which the requester asks for a requester access token. (The AM may also choose to issue a refresh token.) It will use this token to gain access to the other authorization API endpoint.

permission endpoint The endpoint at which the requester asks for authorization to have a new permission associated with its requester access token.

Finally, the host presents one or more protected resource endpoints to the requester:

protected resource endpoint An endpoint at which a requester attempts to access resources. This can be a singular API endpoint, one of a set of API endpoints, a URI corresponding to an HTML document, or any other URI. The requester needs to present a requester access token associated with sufficient permissions in order to gain access.

Similarly to OAuth authorization servers, an UMA AM has the opportunity to manage the validity periods of the access tokens, the corresponding refresh tokens, and even the client credentials that it issues. Different lifetime strategies may be suitable for different resources and scopes of access, and the AM has the opportunity to give the authorizing user control through policy.

Access tokens are currently assumed to be merely opaque strings (as discussed in Section 1.5 and Section 7). Thus, when an AM associates a permission with a requester access token, a host cannot subsequently inspect such a token locally to assess whether a needed permission has been granted. It must instead ask the AM to provide the token's status.

1.4. Scopes, Resource Sets, Permissions, and Authorization

UMA extends the OAuth concept of a "scope" by defining scopes as applying to particular labeled 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. Hosts register resource sets and their scopes without there being a requester in the picture.

Resource sets and scopes have meaning only to hosts and their users, in the same way that application-specific host APIs have meaning only to these entities. The AM is merely a conveyor of labels and descriptions for these constructs, to help the authorizing user set policies that guide eventual authorization processes.

In contrast to an UMA scope, an UMA permission reflects an _actual_ authorization process for a requester to access a particular resource set in a scoped (bounded) manner. Hosts register permission requests on behalf of requesters that have attempted access. Requesters subsequently ask AMs for permissions to be associated with their tokens. AMs grant (or deny) permissions to requesters.

In order to represent meaningful, auditable, and potentially legally enforceable authorization (see [UMA-trustmodel]), a permission is bound to a particular set of UMA entities and parties. This includes the requesting party, the requester (so that the same requesting party would have to go through the authorization process for each client application they use), the host, the resource set on which access is being attempted, and therefore also the AM protecting it and the authorizing user who is controlling access.

Unlike scopes (but similarly to tokens themselves; see Section 1.3), permissions have a validity period.

1.5. AM Metadata

The AM MUST provide an XRD 1.0-formatted document at the hostmeta location (see hostmeta [hostmeta]), documenting the following:

- Major conformance options supported by the AM (described further in Section 7)

- Protection and authorization API endpoints (as described in Section 1.3)

See Section 9 for a full example of AM metadata.

XRD property type values for conformance options:

http://docs.kantarainitiative.org/uma/1.0/client_reg
: OPTIONAL (zero or one). Whether dynamic client registration, such as through [OCDynClientReg], is supported for both hosts and requesters. The only options currently available are "yes" (dynamic registration is supported, using an unspecified method) and "no" (it is not supported; hosts and requesters are required to pre-register). The default is currently AM-specific. (This conformance option is largely a placeholder for now.)

http://docs.kantarainitiative.org/uma/1.0/token_formats
: REQUIRED (one or more). Access token format produced by this AM. Currently the only option defined by this specification is "artifact", meaning an opaque token string as supported natively by the UMA protocol, and the AM is REQUIRED to support this option (and supply this value). The AM MAY provide support for additional access token formats, indicated by extension values that MUST begin with "X-" or "x-".

http://docs.kantarainitiative.org/uma/1.0/claim_formats
: OPTIONAL (zero or more). Claim formats and associated claims-gathering sub-protocols supported by this AM. Currently the only option defined by this specification is "openid", for which details are supplied in Section 3.6.1.1. The AM MAY provide support for additional claim formats, indicated by extension values that MUST begin with "X-" or "x-". If the AM is capable of requesting and accepting any claim formats at all, it MUST declare them.

XRD link relationship rel values for protection API endpoints:

http://docs.kantarainitiative.org/uma/1.0/host_token_uri
: REQUIRED. The host access token endpoint. Available HTTP methods are as defined by [OAuth2] for a token endpoint. Supplies the endpoint the host uses to ask for a host access token.

http://docs.kantarainitiative.org/uma/1.0/host_user_uri
: REQUIRED. The host user authorization endpoint. AAvailable HTTP methods are as defined by [OAuth2] for an end-user authorization endpoint. Supplies the endpoint the host uses to

gather the consent of the authorizing user for a host-AM relationship if it is using the authorization code grant type. The AM MUST support the authorization code grant type method of obtaining the authorizing user's consent.

http://docs.kantarainitiative.org/uma/1.0/host_resource_reg_uri
: REQUIRED. The resource set registration endpoint. Requests to this endpoint require a host access token to be present. Supplies the endpoint the host uses for registering resource sets with the AM to be protected (see Section 2.4.3). This endpoint SHOULD require the use of a transport-layer security mechanism such as TLS.

http://docs.kantarainitiative.org/uma/1.0/host_token_status_uri
: REQUIRED. The token status endpoint. Requests to this endpoint require a host access token to be present. Supplies the endpoint the host uses to request the status of access tokens presented to them by requesters with respect to currently valid permissions. This endpoint SHOULD require the use of a transport-layer security mechanism such as TLS.

http://docs.kantarainitiative.org/uma/1.0/host_perm_reg_uri
: REQUIRED. The permission registration endpoint. Requests to this endpoint require a host access token to be present. Supplies the endpoint the host uses for registering permissions with the AM for which a requester will be seeking authorization (see Section 3.4). This endpoint SHOULD require the use of a transport-layer security mechanism such as TLS.

XRD link relationship rel values for authorization API endpoints:

http://docs.kantarainitiative.org/uma/1.0/req_token_uri
: REQUIRED. The requester access token endpoint. Available HTTP methods are as defined by [OAuth2] for a token issuance endpoint. Supplies the endpoint the requester uses to ask for an access token. This endpoint SHOULD require the use of a transport-layer security mechanism such as TLS.

http://docs.kantarainitiative.org/uma/1.0/req_perm_uri
: REQUIRED. The permission endpoint. Supplies the endpoint the requester uses to ask for authorization to have a new permission associated with its existing requester access token, which MUST accompany the request. This endpoint SHOULD require the use of a transport-layer security mechanism such as TLS.

2. Protecting a Resource

Phase 1 of UMA is protecting a resource. For a host to be able to delegate authorization of protected-resource access to an AM, the authorizing user must first introduce the host to the AM. This phase is concluded successfuly when:

- The host has received metadata about the AM, such as endpoints it needs to use in interacting with the AM.
- The host has received an OAuth host access token that represents the authorizing user's approval for the host to work with the AM in protecting resources. This host access token is later used when the host makes other requests at the AM's protection API.
- The AM has acquired information about resource sets on the host it is supposed to protect on behalf of the authorizing user.

The user, host, and AM perform the following steps in order to successfully complete Phase 1:

1. The host looks up the AM's metadata and learns about its protection API endpoints and supported formats.
2. If the host has not yet obtained a unique OAuth client identifier and optional secret from the AM, it registers with the AM as required. It MAY do this using [OCDynClientReg], if the AM supports it.
3. The host obtains a host access token from the AM with the authorizing user's consent, using either the authorization code grant type or the SAML bearer assertion grant type.
4. The host registers any resource sets with the AM that are intended to be protected.

2.1. Host Looks Up AM Metadata

The host needs to learn the AM's protection API endpoints before they can begin interacting. The authorizing user might provide the AM's location to get the host started in this process, for example by typing a URL into a web form field or clicking a button. Alternatively, the host might already be configured to work with a single AM without requiring any user input. The exact process is beyond the scope of this specification, and it is up to the host to choose a method to learn the AM's location.

From the data provided, discovered, or configured, the host MUST use

the process described in Section 2 of hostmeta [hostmeta] to retrieve the AM hostmeta document. For example, if the user supplied "am.example.com" as the Authorization Manager's domain, the host creates the URL "https://am.example.com/.well-known/host-meta" and performs a GET request on it. The AM MUST return content that includes UMA protection API endpoints as defined in Section 1.5 (see Section 9 for an example).

2.2. Host Registers with AM

If the host has not already obtained a unique client identifier and optional secret from this AM, in this step it MUST do so in order to engage in OAuth-based interactions with the AM. It MAY do this using [OCDynClientReg], if the AM supports it (see Section 1.5 for how the AM MAY indicate support). The AM MUST issue a unique client identifier to every host. This is to ensure that individual hosts can be unambiguously identified in resource set registration, where the client identifier is used as a URI component.

2.3. Host Obtains Host Access Token

In this step, the host acquires a host access token from the AM that represents the approval of the authorizing user for the host to trust the AM for protecting resources belonging to the user.

The host MUST use the OAuth2 [OAuth2] authorization code grant type or the SAML bearer token grant type [OAuth-SAML], utilizing the end-user authorization and token endpoints as appropriate. Here the host acts in the role of an OAuth client; the authorizing user acts in the role of an OAuth end-user resource owner; and the AM acts in the role of an OAuth authorization server. (Once the host has obtained an access token, it presents it to the AM at various protection API endpoints, at which point the AM acts in the role of a resource server.)

The host has completed this step successfully when it possesses a host access token it can use at the AM's protection API.

2.4. Host Registers Sets of Resources to Be Protected

Once the host has received a host access token, for any of the user's sets of resources that are to be protected by this AM, it MUST register these resource sets at the AM's registration endpoint.

Note that the host is free to offer the option to protect any subset of the user's resources using different AMs or other means entirely, or to protect some resources and not others. Additionally, the choice of protection regimes can be made explicitly by the user or

implicitly by the host. Any such partitioning by the host or user is outside the scope of this specification.

See Section 10 for an extended example of registering resource sets.

2.4.1. Scope Descriptions

The host defines a scope of access that is available for use with resources it manages in a document accessible to the AM that contains a scope description. The scopes available for use at any one host MUST have unique URI references so that the host's scope descriptions are distinguishable by URI reference; the URI reference MAY include a fragment identifier. Scope descriptions MAY reside anywhere; the host is not required to self-host scope descriptions and may wish to point to standardized scope descriptions residing elsewhere. (See Section 1.4 for further discussion of scope-related concepts, and Section 10 for a long-form example of scopes used in resource set registration.)

A scope description is a JSON [RFC4627] object with the name "scope" and with the following parameters:

_id REQUIRED. A string that uniquely identifies the scope across all scopes available at this host.

name REQUIRED. A human-readable string describing the scope of access. The AM SHOULD use the name in its user interface to assist the user in setting policies for protected resource sets that have this available scope.

icon_uri OPTIONAL. A URI for a graphic icon representing the scope. If this is provided, the AM SHOULD use the referenced icon in its user interface to assist the user in setting policies for protected resource sets that have this available scope.

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

```
{
        "scope":
            {
            "_id": "view"
            "name": "Read-only",
            "icon_uri": "http://www.example.com/icons/reading-glasses"
        }
}
```

Scope descriptions MAY contain extension parameters that are not

defined in this specification. The names of extension parameters MUST begin with "x-" or "X-".

2.4.2. Resource Set Descriptions

The host defines a resource set that needs protection by registering a resource set description at the AM. The host registers the description and manages its lifecycle at the AM's host resource set registration endpoint by using the resource set registration API, as defined in Section 2.4.3.

The resource set description is a JSON [RFC4627] object with the name "resource_set" and with the following parameters:

_id REQUIRED. A string that uniquely identifies the resource set. The resource set identifier has meaning only to the host. The AM merely maps this resource set description to a particular user by reference to the host access token that was used to access the resource set registration API. The host MAY use any identifier scheme to represent resource sets, for example, making its identifiers unique across all users of this host or allowing for the sharing of resource set identifiers among users. However, for privacy reasons, it is RECOMMENDED that the host assign an identifier that is obscured with respect to any human-readable resource set label used at this host. Further, this identifier MUST match the resource set identifier path component of the URI used to manage this description in the resource set registration API; see Section 2.4.3 for more information. (Typically this matching is achieved through automatically populating the parameter value on initial registration of the description.)

name REQUIRED. A human-readable string describing a set of one or more resources. The AM SHOULD use the name in its user interface to assist the user in setting policing for protecting this resource set.

icon_uri OPTIONAL. A URI for a graphic icon representing the resource set. If provided, the AM SHOULD use the referenced icon in its user interface to assist the user in setting policies for protecting this resource set.

scopes REQUIRED. An array referencing one or more URI references of scope descriptions that are available for this resource set.

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 scopes descriptions as defined in Section 2.4.1:

```
{
        "resource_set":
            {
            "_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"]
        }
}
```

Resource set descriptions MAY contain extension parameters that are not defined in this specification. The names of extension parameters MUST begin with "x-" or "X-".

When a host creates or updates a resource set description (see Section 2.4.3), the AM MUST attempt to retrieve the referenced scope descriptions. It MAY cache such descriptions as long as indicated in the HTTP cache-control header for the scope description resource unless the resource set description is subsequently updated within the validity period. At the beginning of an authorizing user's login session at the AM, the AM MUST attempt to re-retrieve scope descriptions applying to that user whose cached versions have expired.

2.4.3. Resource Set Registration API

The host uses a RESTful API at the AM's resource set registration endpoint to create, read, update, and delete resource set descriptions, along with listing groups of such descriptions. The host MUST use its valid host access token obtained previously to gain access to this endpoint.

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

The AM MUST present an API for registering resource set descriptions at a set of URIs with this structure: "{rsreguri}/host/{hostid}/resource_set/{rsid}"

The components of these URIs are defined as follows:

{rsreguri} The AM's resource set registration endpoint as advertised in its metadata (see Section 1.5).

{hostid} A registration area at the AM that is specific to this host. The host MUST use the unique OAuth client identifier it was assigned by this AM as its host identifier. If the host identifier does not match the host access token used at the host registration endpoint, the AM MUST report an HTTP 403 Forbidden error and fail to act on the request.

{rsid} An identifier for a resource set description. The identifier MUST match the "_id" parameter value in the description itself.

Without a specific resource set identifier path component, the URI applies to the set of resource set descriptions already registered.

Following is a summary of the five registration operations the AM is REQUIRED to support. Each is defined in its own section below. All other methods are unsupported.

- Create resource set description: PUT /host/{hostid}/resource_set/{rsid}
- Read resource set description: GET /host/{hostid}/resource_set/{rsid}
- Update resource set description: PUT /host/{hostid}/resource_set/{rsid}
- Delete resource set description: DELETE /host/{hostid}/resource_set/{rsid}
- List resource set descriptions: GET /host/{hostid}/resource_set/

If the request to the resource set registration endpoint is incorrect, then the AM responds with an error message (see Section 4.2) by including one of the following error codes with the response:

unsupported_method_type The host request used an unsupported HTTP method. The AM MUST respond with the HTTP 403 (Forbidden) status code and MUST fail to act on the request.

hostid_access_token_mismatch The hostid does not match the presented host access token. The AM MUST respond with the HTTP 403 (Forbidden) status code.

ambiguous_resource_set_id The resourcesetid provided in the resource set description does not match the one provided in the URI. The AM MUST respond with the HTTP 400 (Bad Request) status code and MUST fail to act on the request.

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

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

For example:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Cache-Control: no-store
{
   "error":"unsupported_method_type"
}
```

2.4.3.1. Create Resource Set Description

Adds a new resource set description using the PUT method, thereby putting it under the AM's protection. The host is free to use its own methods of identifying and describing resource sets; the AM MUST treat them as opaque for the purpose of authorizing access, other than associating them with the authorizing user represented by the host access token used to access the API. On successfully registering a resource set, the host MUST use UMA mechanisms to limit access to any resources corresponding to this resource set, relying on the AM to supply currently valid permissions for authorized access.

HTTP request:

```
PUT /host/{hostid}/resource_set/{rsid} HTTP/1.1
Content-Type: application/json
...

(Body contains JSON representation of resource set description to be created)
```

Example of an HTTP request that creates a resource set description at the AM:

```
PUT /host/photoz.example.com/resource_set/112210f47de98100 HTTP/1.1
Content-Type: application/json
Host: am.example.com

{
        "resource_set":
            {
            "_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"]
        }
}
```

HTTP response (success):

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: (URL of created resource, same as in the PUT request)
ETag: (entity tag of resource artifact)
...

(Body contains JSON representation of created resource set description)
```

Example of an HTTP response confirming the created resource set description:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://am.example.com/rsreg_uri/host/photoz.example.com/
resource_set/112210f47de98100
ETag: "1234sdbdDX"
...

{
        "resource_set":
            {
            "_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"]
        }
}
```

2.4.3.2. Read Resource Set Description

Reads a previously registered resource set description using the GET method.

```
HTTP request:
GET /host/{hostid}/resource_set/{rsid} HTTP/1.1
...
```

Example of an HTTP request that reads a resource set description from the AM:

```
GET /host/photoz.example.com/resource_set/112210f47de98100 HTTP/1.1
Host: am.example.com
...
```

```
HTTP response (success):
HTTP/1.1 200 OK
Content-Type: application/json
ETag: (entity tag of resource artifact)
...

(Body contains JSON representation of resource set description)
```

Example of an HTTP response message containing a resource set description from the AM:

```
HTTP/1.1 200 OK
Content-Type: application/json
ETag: "1234sdbdDX"
...

              {
        "resource_set":
            {
            "_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"]
        }
```

```
   HTTP response (not found):
   HTTP/1.1 404 Not Found
   Content-Type: application/json
   ...

   {
      "error":"resource_set_not_found"
   }
```

2.4.3.3. Update Resource Set Description

Updates a previously registered resource set description using the PUT method, thereby changing the resource set's protection characteristics.

This operation is different from the operation to create a new resource set description (Section 2.4.3.1) because it assumes that prior registration of the resource set in question has occurred.

HTTP request:

```
PUT /host/{hostid}/resource_set/{rsid} HTTP/1.1
Content-Type: application/json
If-Match: (entity tag of resource if operation is to be idempotent)
...

(Body contains JSON representation of resource set description to be updated)
```

Example of an HTTP request that updates a resource set description at AM:

```
   PUT /host/photoz.example.com/resource_set/112210f47de98100 HTTP/1.1
   Content-Type: application/json
   Host: am.example.com
   If-Match: "1234sdbdDX"

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

```
   HTTP response (success):
   HTTP/1.1 204 No Content
                ETag: "54223dfda"
   ...

   HTTP response (entity tag does not match):
   HTTP/1.1 412 Precondition failed
   Content-Type: application/json
   ...

   {
      "error":"resource_set_mismatch"
   }
```

2.4.3.4. Delete Resource Set Description

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

```
   HTTP request:
   DELETE /host/{hostid}/resource_set/{rsid}
   If-Match: (entity tag of resource if operation is to be idempotent)
   ...
```

Example of an HTTP request that deletes a resource set description from the AM:

```
DELETE /host/photoz.example.com/resource_set/112210f47de98100 HTTP/1.1
Host: am.example.com
If-Match: "1234sdbdDX"
```

```
   HTTP response (success):
   HTTP/1.1 204 No content
   ...

   HTTP response (not found):
   HTTP/1.1 404 Not Found
   Content-Type: application/json
   ...

   {
      "error":"resource_set_not_found"
   }
```

```
HTTP response (entity tag does not match):
HTTP/1.1 412 Precondition failed
Content-Type: application/json
...

{
   "error":"resource_set_mismatch"
}
```

2.4.3.5. List Resource Set Descriptions

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

```
HTTP request:
GET /host/{hostid}/resource_set HTTP/1.1
...
```

```
Example of an HTTP request that lists registered resource set
descriptions at the AM:
GET /host/photoz.example.com/resource_set HTTP/1.1
Host: am.example.com
...
```

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

(Body contains JSON array of {rsid} values)
```

```
Example of an HTTP response with the list of registered resource set
identifiers:
HTTP/1.1 200 OK
Content-Type: application/json
...

{
  "resource_set_id_list": [ "112210f47de98100", "34234df47eL95300" ]
}
```

3. Getting Authorization and Accessing a Resource

Phase 2 of UMA is getting authorization, and Phase 3 is accessing a resource. In these phases, an AM orchestrates and controls requesting parties' access to a user's protected resources at a host,

under conditions dictated by that user.

Phase 3 is merely the successful completion of a requester's access attempt (see Section 3.1.5) that initially involved several embedded interactions among the requester, AM, and host in Phase 2. Phase 2 always begins with the requester attempting access at a protected resource endpoint at the host. How the requester came to learn about this endpoint is out of scope for UMA; the authorizing user might, for example, have advertised its availability publicly on a blog or other website, listed it in a discovery service, or emailed a link to a particular intended requesting party.

The host responds to the requester's access request in one of several ways depending on the circumstances of the request, either immediately or having first performed one or more embedded interactions with the AM. Depending on the nature of the host's response to an failed access attempt, the requester itself engages in embedded interactions with the AM before re-attempting access.

The interactions are as follows. The interaction summarized in each top-level list item MAY be the last interaction engaged in, if the requester chooses not to continue pursuing access to the resource.

- The requester attempts access at a particular protected resource at a host (see Section 3.1).
  - If the user corresponding to the protected resource URI is ambiguous: host responds immediately with an error (see Section 3.1.1).
  - If the user is unambiguous but the access attempt is unaccompanied by a requester access token: host responds immediately with instructions on where to go to obtain one (see Section 3.1.2).
- If the access attempt was accompanied by a requester access token, the host checks the token's status at the AM (see Section 3.3).
  - If the AM reports that the requester access token is invalid (see Section 3.3.2), the host responds to the requester with instructions on where to go to obtain a token (see Section 3.1.2).
- If the AM supplies a token status description for a valid requester access token (see Section 3.3.1) but none of the permissions associated with the token match the scope of attempted access, the host registers a suitable permission on the requester's behalf at the AM (see Section 3.4) and then responds

to the requester with instructions on where to go to request authorization to associate that permission with its token (see Section 3.1.4).

- If the requester received instructions on where to get a token, it requests a token from the appropriate AM (see Section 3.2).

- If the requester received instructions on where to get authorization for access permission, it requests permission from the appropriate AM (see Section 3.5).

- If the AM gave status back on a valid requester access token, and at least one of the permissions associated with the token match the scope of attempted access, the host responds to the requester's access attempt with success (see Section 3.1.5).

The interactions are described in detail in the following sections.

3.1. Requester-Host: Attempt Access at Protected Resource

This interaction assumes that the host has previously registered with an AM one or more resource sets that correspond to the resource to which access is being attempted, such that the host considers this resource to be protected by a particular AM.

The requester typically attempts to access the desired resource at the host directly (for example, when a human operator of the requester software clicks on a thumbnail representation of the resource). The requester is expected to discover, or be provisioned with, knowledge of the protected resource and its location out of band. Further, the requester is expected to acquire its own knowledge about the methods made available by the host for operating on this resource (such as viewing it with a GET method, or transforming it with some complex API call) and the possible scopes of access.

The host responds in one of five ways.

3.1.1. Requester's Request Is Ambiguous

By the nature of the requester's request for access (for example, through a URI parameter specifying a username or other identifier), the host needs to be able to detect uniquely which one of its users has the operative control over access to this resource. Without this, the host will be unable to interact with the correct AM using the correct host access token in protecting the resource.

If the requester's request is ambiguous with respect to the specific

user at the host, the host immediately responds with an "ambiguous-user" error message (see Section 4.2).

For example:
```
HTTP/1.1 403 Forbidden
Content-Type: application/json
Cache-Control: no-store

{
   "error":"ambiguous-user"
}
```

3.1.2. Requester Presents No Access Token

If the host is able to detect uniquely which one of its users has the operative control over access to the resource (see Section 3.1.1), but the requester does not present any access token with the request, the host MUST return an HTTP 400 (Bad Request) status code indicating it is an "invalid_request" (see Section 2.4.1 of [OAuth-bearer]), along with providing the AM's URI. This error indicates to the requester that the request is missing a required parameter, includes an unsupported parameter or parameter value, repeats the same parameter, uses more than one method for including an access token, or is otherwise malformed.

For example:
```
HTTP/1.1 400 Bad Request
WWW-Authenticate: UMA realm="example"
                     host_id="photoz.example.com",
                     am_uri="http://am.example.com"
```

3.1.3. Requester Presents an Invalid Access Token

If the requester presents an access token with its request, the host asks the AM to give it the requester access token's status (see Section 3.3). If the AM reports that the token is invalid, the Host MUST return an HTTP 401 (Unauthorized) status code indicating it is an "invalid_token" (see Section 2.4.1 of [OAuth-bearer]), along with providing the AM's URI.

For example:
```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="example"
                     host_id="photoz.example.com",
                     am_uri="http://am.example.com"
```

3.1.4. Requester's Token Has Insufficient Permission

If the requester presents an access token with its request, the host SHOULD ask the AM to give it the requester access token's status (see Section 3.3). If the AM supplies a token status description for a valid requester access token, the host examines the token status description. If the token status is not associated with any currently valid permission that applies to the scope of access attempted by the requester, the Host SHOULD register the desired permission with the AM (see Section 3.4) and then respond to the requester with the HTTP 403 (Forbidden) status code indicating that the token has "insufficient_scope" (see Section 2.4.1 of [OAuth-bearer]), along with providing the AM's URI and the permission ticket it just received from the AM.

For example:

```
HTTP/1.1 403 Forbidden
WWW-Authenticate: UMA realm="example"
                    host_id="photoz.example.com"
                    am_uri="http://am.example.com",
                    ticket="5454345rdsaa4543"
```

3.1.5. Requester's Token Has Sufficient Permission

If the requester presents an access token with its request, the host SHOULD ask the AM to give it the requester access token's status (see Section 3.3) If the AM supplies a token status description for a valid requester access token, the host examines the token status description. If the token status is associated with at least one currently valid permission that applies to the scope of access attempted by the requester, the host SHOULD give access to the desired resource.

For example:

```
HTTP/1.1 200 OK
Content-Type: image/jpeg
...

/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja
3kAAQAEAAAAPAAA/+4ADkFkb2JlAGTAAAAAAf
/bAIQABgQEBAUEBgUFBgkGBQYJCwgGBggLDAo
KCwoKDBAMDAwMDAwQDA4PEA8ODBMTFBQTExwb
```

This response constitutes the conclusion of Phase 3 of UMA.

3.2. Requester-AM: Requester Obtains Access Token

When a requester does not possess a valid access token to access resources of a particular user at a particular host, it requests one from the AM's requester token endpoint.

The requester learns about this endpoint by retrieving the AM's hostmeta document (see Section 1.5) based on the "am_uri" information that was provided by the host in its previous response, as described in Section 2 of hostmeta [hostmeta]. For example, if the "am_uri" is "am.example.com", the requester creates the URI "https://am.example.com/.well-known/host-meta" and performs a GET request on it.

Each such token represents the set of permissions for that requesting party to access potentially many different resource sets (all controlled by a single authorizing user), with a variety of scopes, at that same host, on behalf of the same requesting party.

The requester SHOULD use the OAuth 2.0 client credentials authorization grant type (see Section 4.4 of [OAuth2]).

In UMA, unlike in plain OAuth, obtaining an access token does not automatically convey permission for access to any protected resource. The token must first be associated with at least one suitable permission for scoped access in order for the requester to succeed in accessing the resource.

If the requester does not yet have a client identifier and optional client secret, it MAY request these using [OCDynClientReg], if the AM supports it (see Section 1.5 for how the AM MAY indicate support).

3.3. Host-AM: Ask for Requester's Presented Access Token Status

On receiving a requester access token in an access attempt, the host asks the AM for the token's status. If it has a cached token status description available that has not expired yet, it MAY use it instead.

The host makes the request to the AM with a POST to the AM's token status endpoint. The body of the HTTP request message contains a JSON [RFC4627] document providing the requester access token and the IP address of the requester's request. The host MAY, at its discretion, instead supply the originating IP address indicated in the requester's X-Forwarded-For: header value. The IP address or originating IP address is advisory only; the AM MAY ignore it for purposes of its own token validation process.

The host gains access to the token status endpoint by presenting its own host access token in the request. The host access token also allows the host and AM to uniquely identify the user they have in common, and therefore allows the AM to look up the correct authorizing user's policies and settings.

Example of a request to the token validation endpoint that provides the host access token in the header:

```
POST /token_status HTTP/1.1
Host: am.example.com
Authorization: Bearer vF9dft4qmT
Content-Type: application/json

{
   "token":"sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv"
   "resource_set_id":"112210f47de98100"
   "host_id":"photoz.example.com"
   "ipaddr":"192.168.1.1"
}
```

3.3.1. AM Returns a Token Status Description

If the the AM finds the requester's access token to be valid, it returns the token's status in an HTTP response using the 200 OK status code, containing a JSON [RFC4627] document supplying the token status description. The token status description contains all of the permissions that are currently valid for this requester access token (and thus for the requesting party on whose behalf it is acting). The AM MAY set a cache period for the returned token status description that allows the host to reuse it over some period of time when it later sees the same requester access token.

The token status description is a JSON object with the name "token_status" containing an array of zero or more permission objects, each with the following parameters:

resource_set_id REQUIRED. A string that uniquely identifies the resource set, access to which has been granted to this requester on behalf of this requesting party. The identifier MUST correspond to a resource set that was previously registered as protected.

scopes REQUIRED. An array referencing one or more URIs of scopes to which access was granted for this resource set. Each scope MUST correspond to a scope that was registered by this host for the referenced resource set.

exp REQUIRED. An integer representing the expiration time on or after which the permission MUST NOT be accepted for authorized access. The processing of the exp parameter requires that the current date/time MUST be before the expiration date/time listed in the exp claim. Host implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew.

Example:

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

{
        "token_status":
           [
              {
                      "resource_set_id": "112210f47de98100",
                      "scopes":
                      ["http://photoz.example.com/dev/actions/view",
                        "http://photoz.example.com/dev/actions/all"],
                      "exp": 1300819380
              }
           ]
}
```

3.3.2. AM Returns a Token Invalid Response

If the the AM finds the requester's access token to be invalid, it returns an UMA error message.

The AM includes one of the following error codes in the error response: (see Section 4.2) and responds with the HTTP 400 status code:

invalid_requester_token AM determined that the requester access token was not valid.

expired_requester_token AM determined that the requester access token has expired.

```
For example:
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
{
   "error":"invalid_requester_token"
}
```

3.4. Host-AM: Register a Permission

If the permissions returned by the AM from a token status request are insufficient to allow this requester's access attempt, the host registers a permission with the AM that it believes would be sufficient for the type of access sought. As a result of the host registering a permission to the AM, the AM returns a permission ticket for the host to give to the requester in its response (see Section 3.1.4).

The permissions ticket is a short-lived opaque structure whose contents is determined by the AM. Later when the requester asks the AM to add permissions to the requester's token (see Section 3.5 it will submit this ticket to the AM. It is therefore the task of the AM to perform binding of this ticket to the requester and its token.

The host registers the permission using the POST method at the AM's permission registration endpoint, providing its host access token to get authorized access to this endpoint. The body of the HTTP request message contains a JSON [RFC4627] document providing the requester's access token and the requested permission.

The requested scope is an object with the name "requested_permission" and the following parameters:

resource_set_id REQUIRED. A string that uniquely identifies a resource set, access to which this requester is seeking access. The identifier MUST correspond to a resource set that was previously registered as protected.

scopes REQUIRED. An array referencing one or more identifiers of scopes to which access is needed for this resource set. Each scope identifier MUST correspond to a scope that was registered by this host for the referenced resource set.

Example of an HTTP request that registers a permission at the AM's permission registration endpoint:

```
POST /host/scope_reg_uri/photoz.example.com HTTP/1.1
Content-Type: application/json
Host: am.example.com

{
        "requested_permission":
            {
            "resource_set_id":  "112210f47de98100",
            "scopes": ["http://photoz.example.com/dev/actions/view",
                        "http://photoz.example.com/dev/actions/all"]
        }
}
```

On receiving the scope registration request from the Host, the AM issues a response message that has one of the possible following outputs:

- A permission ticket and its expiration time (typically very short).
- Error message indicating a malformed scope registration request.

3.4.1. AM Returns a Permission Registration Success Response

The AM responds with an HTTP 201 (Created) status code and includes the Location header in its response as well as the "ticket" parameter in the JSON-formatted body:

For example:

```
HTTP/1.1 201 Created
Content-Type: application/json
Location: https://am.example.com/permreg/host/photoz.example.com/
5454345rdsaa4543

{
   "ticket":"5454345rdsaa4543"
}
```

3.4.2. AM Returns a Permission Registration Error Response

The AM responds with an HTTP 400 (Bad Request) status code and includes one of the following error codes with the error response (see Section 4.2):

invalid_resource_set_id The provided resource set identifier was not found at the AM.

invalid_scope At least one of the scopes included in the request was not registered previously by this host.

invalid_requester_token The requester access token was not recognized by the AM.

expired_requester_token The requester access token has expired.

For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
...

{
   "error":"invalid_resource_set_id"
}
```

3.5. Requester-AM: Request Authorization to Add Permission

In this interaction, the requester asks the AM to grant it permission for access. It does this at the AM's permission endpoint by supplying the permission ticket it got from the host, along with its requester access token and other pertinent information. The AM uses the ticket to look up the previously registered permission, uses the token to confirm that the correct requester is asking for it, maps the requested permission to operative user policies, and ultimately responds to the request positively or negatively.

The requester learns about this endpoint by retrieving the AM's hostmeta document (see Section 1.5) based on the "am_uri" information that was provided by the host in its previous response, as described in Section 2 of hostmeta [hostmeta]. For example, if the "am_uri" is "am.example.com", the requester creates the URI "https://am.example.com/.well-known/host-meta" and performs a GET request on it.

The requester performs a GET or POST on the permission endpoint, supplying:

- The permission ticket it received from the host
- Its own requester access token

- A state parameter (to help avoid replay attacks)
- A redirect URL
- A callback URL

The AM MUST support GET requests to this endpoint and MAY support POST requests; if it supports POST, the endpoint MUST use SSL/TLS. (Requesters will tend to prefer POST when they want to sign the request message and preserve certain URL information; however, GET typically provides a smoother user experience.)

The AM responds with a successful or unsuccessful permission response.

3.5.1. AM Returns an Add Permission Success Response

If the AM determines that the requesting party meets the authorization criteria set out by the authorizing user's policy (see Section 3.6), it responds with an HTTP 201 (Created) status code and provides an updated token:

For example:
HTTP/1.1 201 Created
Content-Type: application/json

```
{
   "token":"sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv"
}
```

3.5.2. AM Returns an Add Permission Error Response

If the content-type of the request is not recognized by the AM, the AM MUST produce an HTTP error.

If the request fails due to missing or invalid parameters, or is otherwise malformed, the AM SHOULD inform the requester of the error by sending an HTTP error response.

3.5.2.1. AM Returns an Add Permission OAuth Error Response

If the request fails due to an invalid, missing, or expired requester access token or requires higher privileges at this endpoint than provided by the access token, the AM responds with an OAuth error (see Section 4.1).

```
For example:
HTTP/1.1 401 Unauthorized
   WWW-Authenticate: Bearer realm="example",
                     error="invalid_token",
                     error_description="The access token expired"
```

3.5.2.2. AM Returns an Add Permission UMA Error Response

The AM responds responds using the appropriate HTTP status code (typically 400 or 403), and includes one of the following error codes in the response: (see Section 4.2):

invalid_requester_ticket The provided ticket was not found at the AM. The AM SHOULD respond with the HTTP 400 (Bad Request) status code.

expired_requester_ticket The provided ticket has expired. The AM SHOULD respond with the HTTP 400 (Bad Request) status code.

not_authorized_permission The requester is definitively not authorized for this permission according to user policy. The AM SHOULD respond with the HTTP 403 (Forbidden) status code.

```
For example:
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
...

{
   "error":"expired_requester_ticket"
}
```

3.6. AM-Requester Authorization Flows

The AM MUST base its decisions to add permissions to requester access tokens on user policies. The nature of these policies is outside the scope of UMA, but generally speaking, they can be thought of as either independent of requesting-party features (for example, time of day) or dependent on requesting-party features (for example, whether they are over 18). This latter case requires the requesting party to transmit identity claims to the AM in some fashion.

The process for requesting and providing claims is extensible and may have a variety of dependencies on the type of requesting party (for example, natural person or legal person) and the type of requester application (for example, browser, native app, or autonomously running web service). UMA currently provides a framework for

handling human-driven requester apps and an optional solution for gathering standardized claims from that end-user, and allows for extensions to support other solutions for this use case and other use cases. The AM SHOULD document its claims-handling ability in its XRD metadata through the claim_formats parameter (see Section 1.5). For the business-level and legal implications of different technical authorization flows, see [UMA-trustmodel].

3.6.1. Authorization Flow for Requester Apps Operated by End-Users

A natural person might be operating a requester app (whether a browser or a native app) in one of two typical situations:

- The requesting party is a natural person (for example, a friend of the authorizing user); the requesting party may even be the authorizing user herself.

- The requesting party is a legal person such as a corporation, and the human being operating the requester app is acting as an agent of that legal person (for example, a customer support specialist representing a credit card company).

The AM has a variety of options at this point for satisfying the authorizing user's policy; this specification does not dictate a single answer. For example, the AM could require the end-user requesting party to register for or log in to a local AM account, or fill in a questionnaire, or complete a purchase.

An end-user-driven requester app MUST redirect the end-user requesting party to the AM to complete the process of authorization. If the AM succeeds in adding the requested permission, it MUST redirect the end-user requesting party back to the requester app when reporting success.

3.6.1.1. Gathering Claims from End-Users with OpenID Connect

An AM MAY use OpenID Connect as one means of gathering claims from an end-user requesting party, leveraging OpenID Connect mechanisms to transmit claims from distributed sources. If it supports this option, the AM MUST supply the "openid" value for one of its "claim_formats" parameters in its AM metadata (see Section 1.5 for how to formulate this metadata).

To conform to this option, the AM MUST do the following:

- Serve as a conforming OpenID Relying Party and Claims Client according to [OCStandard]

- Be able to utilize at least all of the reserved claims defined in [OCMessages] in assessing policy and granting permissions

The AM can then use any conforming OpenID Connect mechanisms and typical user interfaces for engaging with the UserInfo endpoints of OpenID Providers and Claims Providers, potentially allowing for the delivery of "trusted claims" (such as a verified email address or a date or birth) on which authorization policy may depend.

4. Error Messages

Ultimately the host is responsible for either granting the access the requester attempted, or returning an error response to the requester with a reason for the failure. [OAuth2] defines several error responses for a resource server to return. UMA makes use of these error responses, but requires the host to "outsource" the determination of some error conditions to the AM. UMA defines its own additional error responses that the AM may give to the host and requester as they interact with it, and that the host may give to the requester.

4.1. OAuth Error Responses

When a client (host or requester) attempts to access one of the AM endpoints Section 1.5 or a client (requester) attempts to access a protected resource at the host, it has to make an authenticated request by including an OAuth access token in the HTTP request as described in [OAuth2] Section 7.

If the client's request failed authentication, the AM or the host responds with an OAuth error message as described throughout Section 2 and Section 3.

4.2. UMA Error Responses

When a client (host or requester) attempts to access one of the AM endpoints Section 1.5 or a client (requester) attempts to access a protected resource at the host, if the client request is successfully authenticated by OAuth means, but is invalid for another reason, the AM or host responds with an UMA error response by adding the following parameters to the entity body of the HTTP response using the "application/json" media type:

error REQUIRED. A single error code. Value for this parameter is defined in the specific AM 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.

Common error codes:

invalid_request The request is missing a required parameter or is otherwise malformed. The AM MUST respond with the HTTP 400 (Bad Request) status code.

For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json
Cache-Control: no-store
...

{
   "error":"invalid_request",
   "error_description":"There is already a resource with this identifier.",
   "error_uri":"http://am.example.com/errors/resource_exists"
}
```

5. Security Considerations

This specification relies mainly on OAuth security mechanisms for protecting the host registration endpoint at the AM so that only a properly authorized host can access it on behalf of the intended user. For example, the host needs to use a valid host access token issued through a user authorization process at the endpoint, and the interaction SHOULD take place over TLS. It is expected that the host will protect its client secret (if it was issued one) and its host access token, particularly if used in "bearer token" fashion.

In addition, this specification dictates a binding between the host access token and the host-specific registration area on the AM to prevent a host from interacting with a registration area not its own.

For information about the technical, operational, and legal elements of trust establishment between UMA entities and parties, which affects security considerations, see [UMA-trustmodel].

6. Privacy Considerations

The AM comes to be in possession of resource set information (such as names and icons) that may reveal information about the user, which the AM's trust relationship with the host is assumed to accommodate. However, the requester is a less-trusted party (in fact, entirely untrustworthy until it acquires permissions for a requester access token in UMA protocol step 2). This specification recommends obscuring resource set identifiers in order to avoid leaking personally identifiable information to requesters through the "scope" mechanism.

For information about the technical, operational, and legal elements of trust establishment between UMA entities and parties, which affects privacy considerations, see [UMA-trustmodel].

7. Conformance

This section outlines conformance requirements for various entities implementing UMA endpoints.

This specification has dependencies on other specifications, as follows:

- OAuth 2.0: AMs, hosts, and requesters MUST support [OAuth2] features named in this specification for conformance. For example, AMs MUST support the authorization code grant type for being introduced to hosts by authorizing users.

- hostmeta: AMs, hosts, and requesters MUST support the [hostmeta] features named in this specification.

- OpenID Connect: AMs MAY support [OCDynClientReg], and MAY choose to conform to the "openid" claim format option corresponding to the OpenID Connect RP role and support for OpenID Connect reserved claims.

The AM's XRD metadata provides a machine-readable method for an AM to indicate certain of the conformance options it has chosen. Several of the metadata fields allow for extensibility. It is RECOMMENDED that UMA developers and deployers document any profiled or extended features formally and use XRD metadata to indicate their usage. See Section 1.5 for information about providing and extending AM metadata.

8. IANA Considerations

BD

9. AM Metadata Example

For example:

```
<!-- Applies to both hosts and requesters -->

<Property
  type="http://docs.kantarainitiative.org/uma/1.0/token_formats">artifact
</Property>
<Property
  type="http://docs.kantarainitiative.org/uma/1.0/claim_formats">openid
</Property>

<!-- Host protection API -->

<!-- AM as authorization server to host-as-client -->
<Link
  rel="http://docs.kantarainitiative.org/uma/1.0/host_token_uri"
  href="https://am.example.com/host/token_uri">
</Link>
<Link
  rel="http://docs.kantarainitiative.org/uma/1.0/host_user_uri"
  href="https://am.example.com/host/user_uri">
</Link>

<!-- AM as resource server to host-as-client -->
<Link
  rel="http://docs.kantarainitiative.org/uma/1.0/host_resource_reg_uri"
  href="https://am.example.com/host/resource_details_uri">
</Link>
<Link
  rel="http://docs.kantarainitiative.org/uma/1.0/host_token_status_uri"
  href="https://am.example.com/host/token_validation_uri">
</Link>
<Link
  rel="http://docs.kantarainitiative.org/uma/1.0/host_perm_reg_uri"
  href="https://am.example.com/host/scope_reg_uri">
</Link>

<!-- Requester authorization API -->

<!-- AM as authorization server to requester-as-client -->
<Link
  rel="http://docs.kantarainitiative.org/uma/1.0/req_token_uri"
  href="https://am.example.com/requester/token_uri">
</Link>

<!-- AM as resource server to requester-as-client -->
<Link
  rel="http://docs.kantarainitiative.org/uma/1.0/req_perm_uri"
  href="https://am.example.com/requester/perm_uri">
</Link>
```

10. Example of Registering Resource Sets

The following example illustrates the intent and usage of resource set registration.

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".

An authorizing user, Alice Adams, has just uploaded a photo of her new puppy to a host, Photoz.example.com, and wants to ensure that this specific photo is not publicly accessible.

Alice has already introduced this host to her AM, CopMonkey.example.com, and thus Photoz has already obtained a host access token from CopMonkey. However, Alice has not previously instructed Photoz to use CopMonkey to protect any other 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 less-draconian policies to those resources. (User experience only. This may have been done at the time Alice introduced the host to the AM, and/or it could have been a global or host-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 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: "viewing" (consisting of various read-only operations) and "all" (consisting of various reading, editing, and printing operations). It defines two Web-accessible JSON-encoded documents called scope descriptions that represent these scopes, which it is able to reuse for all of its users (not just Alice).

The "name" parameter 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 AM

they use. Photoz could distinguish natural-language labels per user if it wishes, by pointing to scopes with differently translated names.)

Example of the "view" scope ,which description is a Web-accessible resource at http://photoz.example.com/dev/scopes/view:

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

Example of the "all" scope, which description is a Web-accessible resource at http://photoz.example.com/dev/scopes/all:

```
{
      "scope":
          {
          "_id": "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" parameter value is intended to be seen by Alice in mapping authorization constraints to specific resource sets and actions when she visits CopMonkey, such that 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 defined just above.

```
{
        "resource_set":
            {
            "_id": "112210f47de98100"
            "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 UMA resource set registration API, presenting its Alice-specific host access token there, to register and assign an identifier to the resource set description. The resource set identifier path component of the URL matches the "_id" parameter value in the description.

```
PUT /host/photoz.example.com/resource_set/112210f47de98100 HTTP/1.1
Content-Type: application/json
...

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

Once this description has been successfully registered, Photoz is responsible for responding to requesters' attempts to access this photo in a way that is consistent with the authorizing user's policies, achieving protection of the resource by "outsourcing" this task to CopMonkey.

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 AM-related tasks (user experience only).

Over time, as Alice uploads other photos and creates and organizes photo albums, and as Photoz makes new action functionality available, 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.

11. Acknowledgments

The current editor of this specification is Thomas Hardjono of MIT. The following people have additionally served as co-authors:

- Paul Bryan, ForgeRock (former editor)
- Domenico Catalano, Oracle Corp.
- Maciej Machulak, Newcastle University
- Eve Maler, XMLgrrl.com
- Lukasz Moren, Newcastle University
- Christian Scholz, COMlounge GmbH (former editor)

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

12. Issues

All issues are now captured at the project's GitHub site (<https://github.com/xmlgrrl/UMA-Specifications/issues>).

13. References

13.1. Normative References

[OAuth-SAML] Campbell, B., "SAML 2.0 Bearer Assertion Grant Type Profile for OAuth 2.0", February 2011, <http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer-03>.

[OAuth-bearer] Jones, M., "The OAuth 2.0 Protocol: Bearer Tokens", June 2011, <http://tools.ietf.org/html/draft-ietf-oauth-v2-bearer-06>.

[OAuth2] Hammer-Lahav, E., "The OAuth 2.0 Protocol", 2010, <http://tools.ietf.org/html/draft-ietf-oauth-v2-16>.

[OCDynClientReg]
Sakimura, N., "OpenID Connect Dynamic Client Registration 1.0", September 2011, <http://openid.net/specs/openid-connect-registration-1_0.html>.

[OCMessages]
Sakimura, N., "OpenID Connect Messages 1.0", September 2011, <http://openid.net/specs/openid-connect-messages-1_0.html>.

[OCStandard]
Sakimura, N., "OpenID Connect Standard 1.0", September 2011, <http://openid.net/specs/openid-connect-standard-1_0.html>.

[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.

[hostmeta]
Hammer-Lahav, E., "Web Host Metadata", May 2011, <http://tools.ietf.org/html/draft-hammer-hostmeta-16>.

13.2. Informative References

[UMA-trustmodel]
Maler, E., "UMA Trust Model", February 2011, <http://kantarainitiative.org/confluence/display/uma/UMA+Trust+Model>.

[UMA-usecases]
Maler, E., "UMA Scenarios and Use Cases", October 2010, <http://kantarainitiative.org/confluence/display/uma/UMA+Scenarios+and+Use+Cases>.

[UMA-userstories]
Maler, E., "UMA User Stories", November 2010, <http://kantarainitiative.org/confluence/display/uma/User+Stories>.

[UMAnitarians]
Maler, E., "UMA Participant Roster", 2011, <http://kantarainitiative.org/confluence/display/uma/Participant+Roster>.

Appendix A. Document History

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

Author's Address

Thomas Hardjono (editor)
MIT

Email: hardjono@mit.edu