Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 30, 2013

T. Hardjono, Ed.
MIT

December 27, 2012

User-Managed Access (UMA) Profile of OAuth 2.0
draft-hardjono-oauth-umacore-06

Abstract

User-Managed Access (UMA) is a profile of OAuth 2.0. UMA defines how resource owners can control access to their protected resources made by clients operated by arbitrary rquesting parties, where the resources reside on any number of resource servers, and where a centralized authorization server governs access based on resource owner 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 June 30, 2013.

Copyright Notice

Table of Contents

1. Introduction

User-Managed Access (UMA) is a profile of OAuth 2.0 [OAuth2]. UMA defines how resource owners can control access to their protected resources made by clients operated by arbitrary requesting parties, where the resources reside on any number of resource servers, and where a centralized authorization server governs access based on resource owner policy. Resource owners configure authorization servers with access policies that serve as implicit authorization grants. Thus, the UMA profile of OAuth includes an authorization grant flow.

UMA serves numerous use cases where a resource owner outsources authorization for access to their resources, potentially even without the run-time presence of the resource owner. A typical example is the following: a web user (an end-user resource owner) can authorize a web app (client) to gain one-time or ongoing access to a protected resource containing his home address stored at a "personal data store" service (resource server), by telling the resource server to respect access entitlements issued by his authorization service (authorization server). The requesting party operating the client might be the resource owner himself, using a web or native app run by an e-commerce company that needs to know where to ship a purchased item, or it might be his friend who is using an online address book service to collect contact information, or it might be a survey company that uses an autonomous web service to compile population demographics. A variety of scenarios and use cases can be found in [UMA-usecases] and [UMA-casestudies].

Practical control of access among loosely coupled parties requires more than just messaging protocols. This specification defines only the technical "contract" between UMA-conforming entities; its companion Binding Obligations specification [UMA-obligations] defines the expected behaviors of parties operating and using these entities. Parties operating entities that claim to be UMA-conforming MUST provide documentation affirmatively stating their acceptance of the binding obligations contractual framework defined in the Binding Obligations specification.

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 resource owner serve as a policy administrator crafting authorization strategies for resources under their control.

The UMA protocol can be considered an advanced profile of [OAuth2]. In order to increase interoperable communication among the authorization server, resource server, and client, it defines several purpose-built APIs related to the outsourcing of authorization, themselves protected by OAuth in embedded fashion.

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

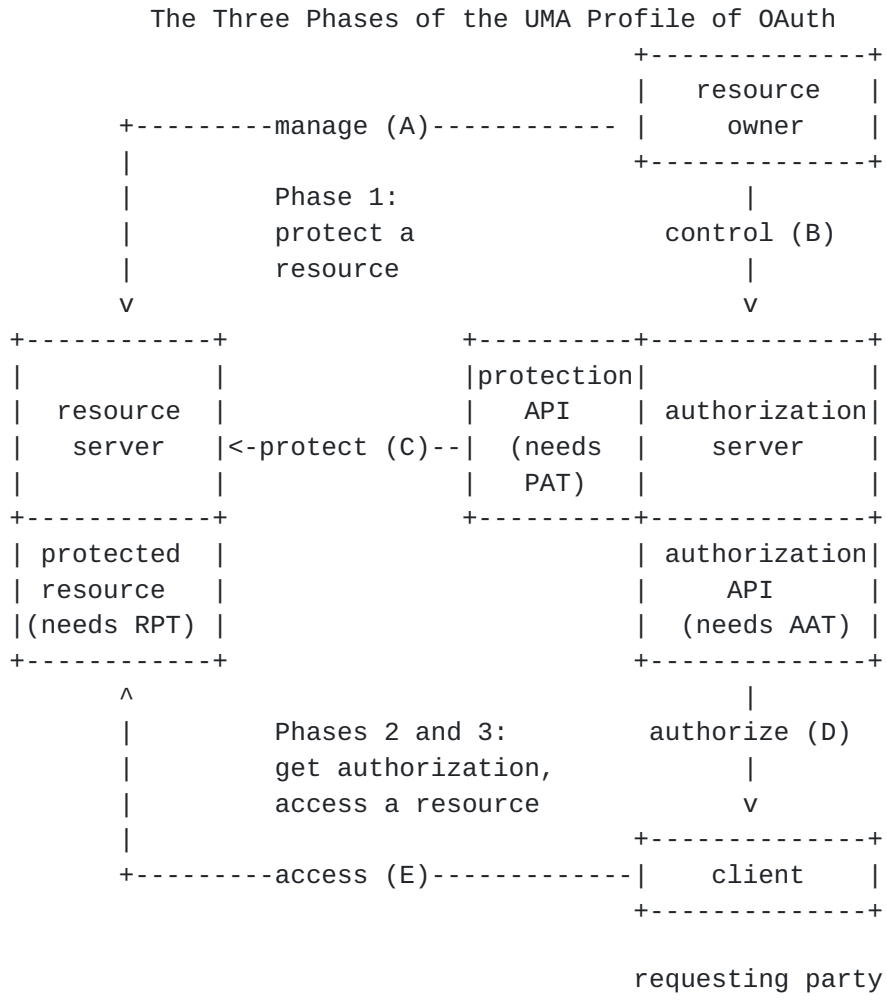

Figure 1

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:_ This phase accomplishes trust establishment among the resource owner, resource server, and authorization server, as well as enabling the resource server to register with the authorization server descriptions of the resources to be protected. This specification uses [OAuth-resource-reg] to enable dynamic introduction and resource registration. In these circumstances, where the resource owner has chosen to use a resource server for managing online resources ("A"), the resource owner introduces this resource server to an authorization server using an OAuth-mediated interaction that results in the authorization server giving the resource server a protection API token (PAT). The resource server then uses the authorization server's protection API to register sets of resources for which protection is being outsourced ("C"). (Out of band of the UMA protocol, the resource owner instructs the authorization server what policies to associated with the registered resource sets ("B").)

2. _Get authorization:_ This phase involves the client (along with its operator, the "requesting party") for the first time. The client approaches the resource server seeking access to a protected resource ("E"). In order to access it, the client must first obtain a requesting party token (RPT) from the authorization server on behalf of its requesting party. The client and requesting party are then redirected to the authorization server to ask for appropriate authorization data (the form of this data depends on the RPT profile in use). In doing so, the requesting party must demonstrate to the authorization server that it satisfies the resource owner's policy governing the sought-for resource and scope ("D"). To use the authorization server's authorization API in the first place, the requesting party has to agree to communication with this server for the purpose of seeking authorization, which results in the client obtaining an authorization API token (AAT).

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

In deploying UMA, implementers are expected to develop one or more profiles of UMA (described in Section 5) that specify and restrict the various UMA protocol options, according to the deployment conditions.

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 properties and values are case sensitive.

1.2. Basic Terminology

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

resource owner
: The "user" in User-Managed Access; an OAuth resource owner. This is typically an end-user (a natural person) but it can also be a corporation or other legal person.

requesting party
: An end-user, or a corporation or other legal person, that uses a client to seek access to a protected resource. The requesting party may or may not be the same party as the resource owner.

client
: An application making protected resource requests with the resource owner's authorization and on the requesting party's behalf.

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 server in order to satisfy policy and gain permission for access to a protected resource.

resource set A set of one or more protected resources. In authorization policy terminology, a resource set is the "object" being protected.

scope type A bounded extent of access that is possible to perform on a resource set. In authorization policy terminology, a scope type is one of the potentially many "verbs" that can logically apply to a resource set ("object"). UMA associates scope types with labeled resource sets.

authorization data Data associated with a requesting party token that enables some combination of the authorization server and resource server to determine the correct extent of access to allow to a client. Authorization data is a key part of the definition of an RPT profile.

permission A scope of access over a particular resource set at a particular resource server that is being asked for by, or being granted to, a requesting party. In authorization policy terminology, a permission is an entitlement that includes a "subject" (requesting party), "verbs" (one or more scopes of access), and an "object" (resource set). A permission is one example of authorization data that an authorization server may grant.

permission ticket A correlation handle that is conveyed from an authorization server to a resource server, from a resource server to a client, and ultimately from a client to an authorization server, to enable the authorization server to assess the correct resource owner policies to apply to a request for an authorization grant.

1.3. Endpoints, Endpoint Protection, and Tokens

Various UMA entities present protected APIs for other entities to use. These APIs are as follows:

- The authorization server presents a _protection API_ to the resource server, which encompasses the resource registration API defined by the OAuth introduction specification [OAuth-resource-reg], as well as additional functions standardized by this specification. This API is OAuth-protected, requiring a resource server to obtain from the authorization server an OAuth access token called a _protection API token (PAT)_.

- The authorization server presents an _authorization API_ to the client, as defined wholly by this specification. This API is OAuth-protected, requiring a client and its requesting party to obtain from the authorization server an OAuth access token, referred to in this specification as an _authorization API token (AAT)_ to distinguish it from other tokens with other purposes.

- The resource server presents a _protected resource_ to the client, which can be considered an application-specific or proprietary API. This API is protected by the UMA profile of OAuth, requiring a client to obtain from the authorization server an OAuth access token, referred to in this specification as a _requesting party token (RPT)_ to distinguish it from other tokens with other

purposes.

The authorization server presents standard OAuth endpoints for token issuance and resource owner authorization in protecting its own UMA APIs, as follows. Resource servers asking to use the protection API would be issued a PAT. Clients and requesting parties asking to use the authorization API would be issued an AAT.

token endpoint Part of standard OAuth, as profiled by UMA. The endpoint at which the resource server asks for a PAT on the resource owner's behalf. Also the endpoint at which the client asks for an AAT on the requesting party's behalf. (The authorization server may also choose to issue a refresh token.) This specification makes the OAuth token profile "bearer" mandatory for the authorization server to implement. It can declare its ability to handle other token profiles.

user authorization endpoint Part of standard OAuth, as profiled by UMA; used when the authorization code grant type (REQUIRED for the authorization server to implement) is being used. The endpoint to which the resource server redirects an end-user resource owner to authorize the former to use this authorization server in outsourcing resource protection. Also the endpoint to which the client redirects the end-user requesting party to authorize the former to use this authorization server in seeking access.

The authorization server presents the following endpoints to the resource server as part of its protection API; these endpoints MUST be OAuth-protected and require a PAT for access, for which the "http://docs.kantarainitiative.org/uma/scopes/prot.json" OAuth scope is required:

resource set registration endpoint The endpoint at which the resource server registers resource sets it wants the authorization server to protect, as defined by [OAuth-resource-reg]. The "http://docs.kantarainitiative.org/uma/scopes/prot.json" scope is a superset of the scope that governs usage of the resource set registration endpoint.

permission registration endpoint The endpoint at which the resource server registers permissions that it anticipates a client will shortly be asking for from the authorization server.

introspection endpoint The endpoint at which the resource server forwards an RPT that has accompanied an access request to learn what authorization data is associated with it, as defined by [OAuth-introspection]. This specification defines an RPT profile, "bearer", which is mandatory for the authorization server to implement and which, if used, REQUIRES the resource server to use this endpoint (see Section 3.3). The "http://docs.kantarainitiative.org/uma/scopes/prot.json" scope is a superset of the scope that governs usage of the token introspection endpoint.

The authorization server presents the following endpoints to the client as part of its authorization API; these endpoints are OAuth-protected and require an AAT for access, for which the "http://docs.kantarainitiative.org/uma/scopes/authorization" OAuth scope is required:

RPT endpoint The endpoint at which the client asks the authorization server for the issuance of an RPT relating to this requesting party, resource server, and authorization server.

permission request endpoint The endpoint at which the client asks for authorization data to be associated with an RPT to enable authorized access.

The resource server presents one or more protected resource endpoints to the client; these endpoints are protected by the UMA profile of OAuth and require an RPT with sufficient authorization data to permit access:

protected resource endpoint An application-specific endpoint at which a client 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 authorization server has the opportunity to manage the validity periods of the access tokens, the corresponding refresh tokens where applicable, and even the client credentials that it issues. Different lifetime strategies may be suitable for different resources and scopes of access, and the authorization server has the opportunity to give the resource owner control through policy. These options are all outside the scope of this specification.

1.4. Scope Types, Resource Sets, Permissions, and Authorization

UMA extends the OAuth concept of a "scope" by defining scope types as applying to labeled resource sets, rather than leaving the relevant resources (such as API endpoints or URIs) implicit. A resource set

can have any number of scope types, 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. Resource servers register resource sets and their scope types when there is not yet any particular requesting party or client in the picture.

Resource sets and scope types have meaning only to resource servers and their users, in the same way that application-specific protected resource APIs have meaning only to these entities. The authorization server is merely a conveyor of labels and descriptions for these constructs, to help the resource owner set policies that guide eventual authorization processes.

A permission, in contrast to a scope type, reflects an _actual_ entitlement to access a resource set using one or more scope types, as the result of an authorization process undergone by a specific requesting party. A resource server registers a permission request with an authorization server on behalf of a client (and its requesting partie) that has attempted access, and transmits the resulting permission ticket to the client. The client subsequently asks the authorization server for authorization data to be associated with its RPT. If the RPT profile is in use, the authorization server grants (or denies) the permission to the requesting party. (If another token profile is in use, the authorization server might generate a different type of authorization data, such as an authorization decision or a package of the claims it has collected.)

An RPT is bound to a requesting party, the client being used by that party, the resource server at which protected resources of interest reside, and the authorization server that protects those resources. It becomes associated with as many pieces of authorization data as are appropriate for gaining authorized access to resources protected at that resource server by any single authorization server (even if that data applies to resources managed by two or more different resource owners at the same resource server using the same authorization server).

In the case of the UMA "bearer" token profile, each individual permission is associated with the resource owner whose policies drove the authorization process. This enables meaningful, auditable, and potentially legally enforceable authorization for access (see [UMA-obligations]). Permissions have a validity period that the authorization server has the opportunity to control (independently or with input from the resource owner). These control options are outside the scope of this specification.

1.5. Authorization Server Configuration Data

The authorization server MUST provide configuration data to other entities it interacts with in aJSON [RFC4627] document that resides in an /uma-configuration directory at at its hostmeta [RFC6415] location. The configuration data documents major conformance options supported by the authorization server (described further in Section 8) and protection and authorization API endpoints (as described in Section 1.3). (At the appropriate time, this section will instead profile whatever self-describing metadata specification OAuth adopts, for example, [OAuth-linktypes] or [OAuth-meta].)

The configuration data has the following properties. All endpoint URIs supplied SHOULD require the use of a transport-layer security mechanism such as TLS.

version
: REQUIRED. The version of the UMA core protocol to which this authorization server conforms. The value MUST be the string "1.0".

issuer
: REQUIRED. A URI indicating the party operating the authorization server.

dynamic_client_endpoint
: OPTIONAL. The endpoint to use for performing dynamic client registration through . [DynClientReg]

oauth_token_profiles_supported
: REQUIRED. PAT and AAT profiles produced by this authorization server. The property value is an array of string values. Currently the only string value for this property defined by this specification is "bearer", corresponding to the OAuth bearer token profile [OAuth-bearer]. The authorization server is REQUIRED to support this profile, and to supply this string value explicitly. The authorization server MAY declare its support for additional access token profiles by providing a unique absolute URI in a string value in the array for each one.

uma_token_profiles_supported
: REQUIRED. RPT types produced by this authorization server. The property value is an array of string values. Currently the only string value for this property defined by this specification is "bearer", whose associations the resource server MUST determine through a token introspection interaction with the authorization server (see Section 3.3 for the

definition of this profile). The authorization server is REQUIRED to support the UMA bearer token profile, and to supply this string value explicitly. The authorization server MAY declare its support for RPTs using additional RPT profiles by providing a unique absolute URI in a string value in the array for each one.

oauth_grant_types_supported
: REQUIRED. OAuth grant types supported by this authorization server in issuing PATs and AATs. The property value is an array of string values. Each string value MUST be one of the grant_type values defined in [OAuth2], or alternatively an extension grant type indicated by a unique absolute URI.

claim_profiles_supported
: OPTIONAL. Claim formats and associated sub-protocols for gathering claims from requesting parties, as supported by this authorization server. The property value is an array of string values. Currently the only string value for this property defined by this specification is "openid", for which details are supplied in Section 3.5.1.1. The authorization server MAY declare its support for additional claim profiles by assigning a unique absolute URI in a string value in the array for each one.

token_endpoint
: REQUIRED. The endpoint URI at which the resource server or client asks the authorization server for a PAT or AAT, respectively. A requested scope of "http://docs.kantarainitiative.org/uma/scopes/prot.json" results in a PAT. A requested scope of "http://docs.kantarainitiative.org/uma/scopes/authorization" results in an AAT. Available HTTP methods are as defined by [OAuth2] for a token endpoint.

user_endpoint
: REQUIRED. The endpoint URI at which the resource server gathers the consent of the end-user resource owner or the client gathers the consent of the end-user requesting party, if the "authorization_code" grant type is used. Available HTTP methods are as defined by [OAuth2] for an end-user authorization endpoint.

permission_registration_endpoint
: REQUIRED. The endpoint URI at which the resource server registers permissions with the authorization server for which a client will be seeking authorization on its requesting party's behalf (see Section 3.2). A PAT MUST accompany requests to

this protected endpoint.

rpt_endpoint
REQUIRED. The endpoint URI at which the client ask the authorization server for an RPT. An AAT token MUST accompany requests to this protected endpoint.

rpt_status_endpoint
REQUIRED. The endpoint URI at which the resource server introspects an RPT presented to it by a client (see Section 3.3). A PAT MUST accompany requests to this protected endpoint.

permission_request_endpoint
REQUIRED. The endpoint URI at which the client asks, on its requesting party's behalf, to have authorization data associated with its RPT. An AAT MUST accompany requests to this protected endpoint.

Example of authorization server configuration data that resides at https://example.com/.well-known/uma-configuration (note the use of https: for endpoints throughout):

```
{
"version":"1.0",
"issuer":"https://example.com",
"dynamic_client_endpoint":"https://as.example.com/dyn_client_reg_uri",
"oauth_token_profiles_supported":[
  "bearer"
],
"uma_token_profiles_supported":[
  "bearer"
],
"oauth_grant_types_supported":[
  "authorization_code"
],
"claim_profiles_supported":[
  "openid"
],
"token_endpoint":"https://as.example.com/token_uri",
"user_endpoint":"https://as.example.com/user_uri",
"resource_set_registration_endpoint":"https://as.example.com/rs/rsrc_uri",
"rpt_status_endpoint":"https://as.example.com/rs/status_uri",
"permission_registration_endpoint":"https://as.example.com/rs/perm_uri",
"rpt_endpoint":"https://as.example.com/client/rpt_uri",
"permission_request_endpoint":"https://as.example.com/client/perm_uri"
}
```

Authorization server configuration data MAY contain extension

properties that are not defined in this specification. Extension names that are unprotected from collisions are outside the scope of the current specification.

2. Protecting a Resource

Phase 1 of UMA is protecting a resource. The resource owner, resource server, and authorization server perform the following steps to successfully complete Phase 1 (assuming that the resource server has discovered the authorization server's configuration data and endpoints as needed):

- The resource server and authorization server establish mutual trust through the issuance of client credentials to the resource server. It is OPTIONAL for the client credentials to be provided dynamically through [DynClientReg]); alternatively, they MAY use a static process.

- The resource owner, resource server, and authorization server establish three-way trust through the issuance of a PAT. See Section 2.1 for additional details.

- The resource server registers any resource sets with the authorization server that are intended to be protected. See Section 2.2 for additional details.

2.1. Resource Server Obtains PAT

In this step, the resource server acquires a PAT from the authorization server. The token represents the approval of the resource owner for this resource server to trust this authorization server for protecting resources belonging to this resource owner. It is OPTIONAL for the resource owner to introduce the resource server to the authorization server dynamically through the process defined in [OAuth-resource-reg]); alternatively, they MAY use a static process that may or may not directly involve the resource owner at introduction time.

The resource server MUST use OAuth 2.0 [OAuth2] to obtain the PAT. Here the resource server acts in the role of an OAuth client requesting the "http://docs.kantarainitiative.org/uma/scopes/prot.json" scope, which authorizes it to use the authorization server's resource set registration endpoint (as defined in [OAuth-resource-reg]) as well as additional protection API endpoints. Once the resource server has obtained its PAT, it presents it to the authorization server at various protection API endpoints.

(NOTE: The "http://docs.kantarainitiative.org/uma/scopes/prot.json" scope keyword is a URI that resolves to a JSON-encoded scope description, in the fashion of UMA scope types. This scope description is non-normative.)

The authorization server MAY support the use of any OAuth grant type for PAT issuance, but MUST support the authorization_code grant type, and SHOULD support the SAML bearer token grant type [OAuth-SAML] (urn:ietf:params:oauth:grant-type:saml2-bearer) if it anticipates working with resource servers that are operating in environments where the use of SAML is prevalent. The authorization server MUST indicate all grant types it supports for PAT issuance in its configuration data.

2.2. Resource Server Registers Sets of Resources to Be Protected

Once the resource server has received a PAT, for any of the resource owner's sets of resources that are to be protected by this authorization server, it registers these resource sets in a timely fashion. To do this, the resource server uses the resource set registration API defined in [OAuth-resource-reg].

Note: The resource server is free to offer the option to protect any subset of the resource owner's resources using different authorization servers or other means entirely, or to protect some resources and not others. Additionally, the choice of protection regimes can be made explicitly by the resource owner or implicitly by the resource server. Any such partitioning by the resource server or owner is outside the scope of this specification.

On successfully registering a resource set, the RS MUST use access control mechanisms to limit access to any resources corresponding to this resource set, relying on the AS to supply currently valid permissions for authorized access. The RS MUST outsource protection to the AS according to the currently registered state of a resource set. This requirement holds true so long as the RS has one or more registsred resource sets.

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 authorization server orchestrates and controls clients' access (on their requesting parties' behalf) to a resource owner's protected resources at a resource server, under conditions dictated by that resource owner.

Phase 3 is merely the successful completion of a client's access

attempt that initially involved several embedded interactions among the client, requesting party, authorization server, and resource server in phase 2. Phase 2 always begins with the client attempting access at a protected resource endpoint at the resource server. How the client came to learn about this endpoint is out of scope for this specification. The resource owner 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 resource server responds to the client'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 authorization server. Depending on the nature of the resource server's response to an failed access attempt, the client and its operator engage in embedded interactions with the authorization server before re-attempting access.

The interactions are as follows. Each interaction MAY be the last, if the client chooses not to continue pursuing the access attempt or the resource server chooses not to continue facilitating it.

1. The client attempts access at a particular protected resource at a resource server (see Section 3.1).

    A. If the access attempt is unaccompanied by an RPT, the resource server responds immediately with an HTTP 401 (Unauthorized) response and instructions on where to go to obtain one (see Section 3.4.2).

    B. If the access attempt was accompanied by an RPT, the resource server checks the RPT's status (see Section 3.3).

        1. If the RPT is invalid (for example, it is not applicable to this resource server), the resource server responds to the client with an HTTP 401 (Unauthorized) response and instructions on where to go to obtain a token (see Section 3.4.2).

        2. If the RPT is valid but has insufficient permission, the resource server registers a suitable permission request on the client's behalf at the authorization server (see Section 3.2), and then responds to the client with an HTTP 403 (Forbidden) response and instructions on where to go to ask for authorization (see Section 3.1.2).

        3. If the RPT is valid, and if the authorization data associated with the token is consistent with allowing

access, the resource server responds to the client's access attempt with an HTTP 200 (OK) response and a representation of the resource (see Section 3.1.3).

2. If the client (possessing no RPT or an invalid RPT) received a 401 response and an RPT endpoint, it then requests an RPT from that endpoint (see Section 3.4.2).

3. If the client (posessing a valid RPT) received a 403 response and a permission ticket, it then asks the authorization server for authorization data that matches the ticket (Section 3.4.3). If the authorization server needs requesting party claims in order to assess this client's authorization, it engages in a claims-gathering flow with the requesting party (see Section 3.5).

   A. If the client does not already have an AAT at the appropriate authorization server to be able to use its permission request endpoint, it first obtains one (see Section 3.4.1).

The interactions are described in detail in the following sections.

3.1. Client Attempts to Access Protected Resource

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

The client typically attempts to access the desired resource at the resource server directly (for example, when an end-user requesting party clicks on a thumbnail representation of the resource). The client is expected to discover, or be provisioned or configured with, knowledge of the protected resource and its location out of band. Further, the client is expected to acquire its own knowledge about the application-specific methods made available by the resource server for operating on this protected resource (such as viewing it with a GET method, or transforming it with some complex API call) and the possible scopes of access.

Example of a request carrying no RPT:

```
GET /album/photo.jpg HTTP/1.1
Host: photoz.example.com
...
```

Example of a request carrying an RPT using the UMA "bearer" token profile:

```
GET /album/photo.jpg HTTP/1.1
Authorization: Bearer vF9dft4qmT
Host: photoz.example.com
...
```

The resource server responds in one of the following ways.

3.1.1. Client Presents No RPT

If the client does not present any access token with the request, the resource server MUST return an HTTP 401 (Unauthorized) status code, along with providing the authorization server's URI in an "as_uri" property to facilitate authorization server configuration data discovery, including discovery of the endpoint where the client can request an RPT (Section 3.4.2).

For example:

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

3.1.2. Client Presents an RPT That Has Insufficient Authorization Data

If the client presents an RPT with its request, the resource server SHOULD determine the RPT's status (see Section 3.3). If the RPT is invalid, the resource server redirects the client to the RPT endpoint at the authorization server to obtain a correct RPT (see Section 3.4.2).

If the RPT is valid but has insufficient permission for the type of access sought, the resource server SHOULD register a permission with the authorization server that would suffice for that scope of access (see Section 3.2), and then respond to the client with the HTTP 403 (Forbidden) status code, along with providing the authorization server's URI in the header of the message and the permission ticket it just received from the authorization server in the body in JSON form.

Example of the host's response:

```
HTTP/1.1 403 Forbidden
WWW-Authenticate: UMA realm="example",
  host_id="photoz.example.com",
  as_uri="http://am.example.com"
  error="insufficient_scope"

{
"ticket": "016f84e8-f9b9-11e0-bd6f-0021cc6004de"
}
```

3.1.3. Client Presents a Valid RPT with Sufficient Authorization Data

If the RPT's status is associated with authorization data that is consistent with authorized access of the scope sought by the client (see Section 3.3), the resource server MUST give access to the desired resource.

Example of the resource server's response:

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

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

This response constitutes the conclusion of [hase 3 of UMA.

The resource server MUST NOT give access where the token's status is not associated with sufficient authorization data for the attempted scope of access.

3.2. Resource Server Registers a Permission With Authorization Server

In response to receiving an access request accompanied by an RPT that is invalid or has insufficient authorization data, the resource server SHOULD register a permission with the authorization server that would be sufficient for the type of access sought. The authorization server returns a permission ticket for the resource server to give to the client in its response.

The permission ticket is a short-lived opaque structure whose form is determined by the authorization server. The ticket value MUST be securely random (for example, not merely part of a predictable

sequential series), to avoid denial-of-service attacks. Since the ticket is an opaque structure from the point of view of the client, the authorization server is free to include information regarding expiration time within the opaque ticket for its own consumption. When the client subsequently asks the authorization server for authorization data to be associated with its RPT, it will submit this ticket to the authorization server.

The resource server registers the permission using the POST method at the authorization server's permission registration endpoint. The resource server MUST provide its valid PAT in order to get access to this endpoint. The body of the HTTP request message contains a JSON document providing the requested permission.

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

resource_set_id REQUIRED. The identifier for a resource set, access to which this client is seeking access. The identifier MUST correspond to a resource set that was previously registered.

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

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

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

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

If the registration request is successful, the authorization server responds with an HTTP 201 (Created) status code and includes the Location header in its response as well as the "ticket" property in the JSON-formatted body.

For example:

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

{
"ticket": "016f84e8-f9b9-11e0-bd6f-0021cc6004de"
}
```

If the registration request is authenticated properly but fails due to other reasons, the authorization server responds with an HTTP 400 (Bad Request) status code and includes one of the following UMA error codes (see Section 4.2):

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

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

3.3. Resource Server Determines the RPT Status

On receiving an RPT, the resource server MUST ascertain its status before granting or denying access to the client. An RPT is associated with a set of authorization data that governs whether the client is authorized for access. The token's nature and format are dictated by its profile; the profile might allow it to be self-contained, such that the resource server is able to ascertain its status locally, or might require or allow the resource server to make a run-time introspection request of the authorization server that issued the token using [OAuth-introspection].

This specification makes one type of RPT mandatory to implement: the UMA bearer token profile, as defined in Section 3.3.1. Alternate RPT profiles MAY define their own unique token formats and MAY require, allow, or prohibit use of the token introspection endpoint.

3.3.1. UMA Bearer Token Profile

This section defines the format and protocol requirements for the UMA bearer token profile. An authorization server MUST support the UMA bearer token profile and MUST indicate its support in the "uma_token_profiles_supported" property in its configuration data (see Section 1.5).

On receiving an RPT of the "Bearer" type in an authorization header

from a client making an access attempt, the resource server MUST use the authorization server's token introspection endpoint [OAuth-introspection] to retrieve the RPT's associated authorization data. In order to ask the authorization server for an RPT's status, the host makes the request to the authorization server with a POST request to the authorization server's token introspection endpoint. The body of the HTTP request message contains a JSON document providing the RPT. The host MUST provide its own PAT in the request in order to gain access to the RPT status endpoint.

Example of a request to the RPT status endpoint that provides the PAT in the header:

```
POST /token_status HTTP/1.1
Host: am.example.com
Authorization: Bearer vF9dft4qmT
...

{
  "rpt": "sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv",
  "resource_set_id": "112210f47de98100",
  "host_id": "photoz.example.com"
}
```

The authorization server returns the RPT's status in an HTTP response using the 200 OK status code, containing a JSON document supplying the RPT's associated permissions. The RPT status description either contains all of the permissions that are currently valid for this RPT or indicates that the RPT is invalid (see Section 1.4). The authorization server MAY set a cache period for the returned RPT status description that allows the host to reuse it over some period of time when it later sees the same RPT.

The status description for a valid RPT is a JSON array of zero or more permission objects, each with the following properties (this needs to be synced up with the token introspection spec):

resource_set_id REQUIRED. A string that uniquely identifies the resource set, access to which has been granted to this client 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 property 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
...

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

The token status description for an invalid RPT is a JSON structure, as follows.

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

{
  "rpt_status": "invalid"
}
```

3.4. Client Asks Authorization Server for RPT and Authorization Data

A client making an access attempt accompanied by no RPT or by an invalid RPT will receive a 401 response back from the resource server, along with the authorization server's location from which it can learn the RPT endpoint. In this case, the client must obtain a valid RPT from the authorization server's RPT endpoint provided in the response (see Section 3.4.2).

A client making an access attempt with a valid RPT that has insufficient authorization data associated with it will receive a 403

response back from the resource server, along with a permission ticket and the authorization server's location from which it can learn the permission request endpoint. In this case, the client uses the permission ticket to ask for the necessary authorization data to be associated with its RPT. This process necessarily involves the requesting party because the authorization is sought on this party's behalf.

The client takes action in the following ways (assuming that it has discovered the authorization server's configuration data and endpoints as required):

- The client and authorization server establish mutual trust through the issuance of client credentials to the client. It is OPTIONAL for the client credentials to be provided dynamically through [DynClientReg]); alternatively, they MAY use a static process.

- The requesting party, client, and authorization server establish three-way trust through the issuance of an AAT. See Section 3.4.1 for additional details.

- The client obtains an RPT. See Section 3.4.2 for additional details.

3.4.1. Client Obtains AAT

In this step, the client acquires an AAT from the authorization server on the requesting party's behalf. The token represents the approval of this requesting party for this client to engage with this authorization server to supply claims, ask for authorization, and perform any other tasks needed for obtaining authorization for access to resources at all resource servers that use this authorization server. It is OPTIONAL for the requesting party to introduce the client to the authorization server dynamically through the process defined in [OAuth-resource-reg]); alternatively, they MAY use a static process that does not directly involve the requesting party.

The client MUST use OAuth 2.0 [OAuth2] to obtain the AAT. Here the client requests the "http://docs.kantarainitiative.org/uma/scopes/authz.json" scope. Once the client has obtained its AAT, it presents it to the authorization server at the permission request endpoint.

(NOTE: The "http://docs.kantarainitiative.org/uma/scopes/authz.json" scope keyword is a URI that resolves to a JSON-encoded scope description, in the fashion of UMA scope types. This scope description is non-normative.)

The authorization server MAY support the use of any OAuth grant type for AAT issuance, but MUST support the authorization_code grant type, and SHOULD support the SAML bearer token grant type [OAuth-SAML] (urn:ietf:params:oauth:grant-type:saml2-bearer) if it anticipates working with clients that are operating in environments where the use of SAML is prevalent. The authorization server MUST indicate all grant types it supports for AAT issuance in its configuration data.

By virtue of being able to identify this client/requesting party pair uniquely across all resource servers whose resources it protects, the authorization server is able to manage the process of authorization and claims-gathering efficiently. These management processes are outside the scope of this specification.

3.4.2. Client Obtains RPT

In this step, if the client needs an RPT that applies to this requesting party for this resource server and this authorization server, it obtains an RPT. On first issuance the RPT is associated with no authorization data and thus does not convey any authorizations for access.

The clieint performs a POST on the RPT endpoint. It MUST provide its own AAT in the header.

Example of a request message containing an AAT:

```
POST /rpt HTTP/1.1
Host: am.example.com
Authorization: Bearer jwfLG53^sad$#f
...
```

The authorization server responds with an HTTP 201 (Created) status code and provides a new RPT.

For example:

```
HTTP/1.1 201 Created
Content-Type: application/uma-rpt+json

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

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

The client might need an RPT if it has never before requested an RPT for this combination of requesting party, resource server, and

authorization server, or if it has lost control of a previously issued RPT and needs a refreshed one. If the AAT provided in the header is the same as one provided for a previously issued RPT by this authorization server, the authorization server invalidates the old RPT and issues a new one.

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

If the request fails due to an invalid, missing, or expired AAT or requires higher privileges at this endpoint than provided by the AAT, the authorization server 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"
```

3.4.3. Client Asks for Authorization Data

Once in possession of an AAT for this authorization server, an RPT that applies to this requesting party for this resource server and this authorization server, and a permission ticket, the client asks the authorization server to give it suitable authorization data for the sought-for access. The client performs a POST on the permission request endpoint, supplying the items below. The client MUST provide its own AAT in the header.

- The permission ticket it received from the resource server
- Its RPT for this resource server
- Its own AAT in the header

Example of a request message containing a permission ticket and RPT:

```
POST /token_status HTTP/1.1
Host: am.example.com
Authorization: Bearer jwfLG53^sad$#f
...

{
 "rpt": "sbjsbhs(/SSJHBSUSSJHVhjsgvhsgvshgsv",
 "ticket": "016f84e8-f9b9-11e0-bd6f-0021cc6004de"
}
```

In this interaction, the client uses the authorization server's permission request endpoint.

The authorization server uses the ticket to look up the details of the previously registered permission, maps the requested permission to operative resource owner policies, undergoes any authorization flows required (see Section 3.5), and ultimately responds to the request positively or negatively. The resource owner's policies at the authorization server amount to an implicit authorization grant in governing the issuance of authorization data. (The authorization server is also free to enable resource owners to set policies that require them to provide a run-time explicit authorization grant mediated by the authorization server. This setting of policies and gathering of authorization grants is outside the scope of this specification.)

If the request fails due to an invalid, missing, or expired AAT (or RPT) or requires higher privileges at this endpoint than provided by the AAT, the authorization server 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"
```

If the authorization server does not add the requested authorization data, it 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 authorization server. The authorization server SHOULD respond with the HTTP 400 (Bad Request) status code.

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

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

need_claims The authorization server is unable to determine whether the client is authorized for this permission without gathering claims from the requesting party. The authorization server SHOULD respond with the HTTP 403 (Forbidden) status code. The client is therefore not authorized, but has the opportunity to engage its operator -- the requesting party -- in a claims-gathering flow with the authorization server (see Section 3.5) to potentially become authorized.

For example:

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

{
  "status": "error",
  "error": "expired_requester_ticket"
}
```

3.5. Claims-Gathering Flows

The authorization server MUST base the addition of authorization data to RPTs 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 client (for example, browser, native app, or autonomously running web service). UMA provides a framework for handling end-user-driven clients and an optional solution for gathering standardized claims from such an end-user, and allows for extensions to support other solutions for this use case and other use cases. The authorization server SHOULD

document its claims-handling ability in its configuration data through the claim_profiles_supported property (see Section 1.5). For the business-level and legal implications of different technical authorization flows, see [UMA-obligations].

3.5.1. Claims-Gathering Flow for Clients Operated by End-Users

A client, whether web-based or native, is operated by an end-user in one of two typical situations:

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

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

For convenience, this specification refers to the end-user as a "requesting end-user" to cover both cases, which differ only at the level of business agreements (and potentially law), rather than technology. The authorization server has a variety of options at this point for satisfying the resource owner's policy; this specification does not dictate a single answer. For example, the authorization server could require the requesting end-user to register for and/or log in to a local authorization server account, or to fill in a questionnaire, or to complete a purchase. It could even require several of these operations, where the order is treated as significant. A variety of claim profiling can be defined to achieve these effects.

An end-user-driven client MUST redirect the requesting end-user to the authorization server to complete the process of authorization. The redirection MUST include a URI query parameter with the name "ticket" whose value conveys the permission ticket for which the need_claims error was received; for example, "ticket=016f84e8-f9b9-11e0-bd6f-0021cc6004de". Each claim profile MUST provide the following capabilities:

redirect URI A means by which the client MUST supply the URI to which the authorization server MUST redirect the requesting end-user at the end of the claims-gathering process.

callback URI A means by which the client OPTIONALLY supplies a callback URI for the authorization server to use.

state A means by which the client SHOULD supply an opaque value used to maintain state between the request and the callback; this serves as a protection against XSRF attacks.

An authorization server MAY support any number of claim profiles. One potential such profile is defined in this specification: the "openid" claim profile, which leverages OpenID Connect for gathering generally useful identity claims (see Section 3.5.1.1).

3.5.1.1. OpenID Connect Claim Profile

This section defines the OpenID Connect claim profile for UMA. Following is a summary:

- Identifying URI: http://docs.kantarainitiative.org/uma/profiles/uma-claim-openid-1.0
- Profile author and contact information: Thomas Hardjono (hardjono@mit.edu)
- Updates or obsoletes: None; this profile is new.
- Authorization server configuration data: To indicate support, supply the keyword "openid" in the "claim_profiles_supported" property value.
- Syntax and semantics of claim data: As defined below. The claim data format leverages the OpenID Connect protocol and the reserved claims defined in that specification.
- Claims gathering method: As defined below.
- Error states: None additional.
- Security and privacy considerations: None additional.
- Binding obligations: Binding obligations that apply to the use of this claim profile are documented in [UMA-obligations].

If an authorization server supports the OpenID Connect claim profile, it MUST supply the "openid" value for one of its "claim_profiles_supported" values in its configuration data.

To conform to this option, the authorization server 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

- Use the OpenID Connect "redirect_uri" and "state" request parameters as appropriate

The authorization server 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 resource server is responsible for either granting the access the client attempted, or returning an error response to the client 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 resource server to "outsource" the determination of some error conditions to the authorization server. UMA defines its own additional error responses that the authorization server may give to the resource server and client as they interact with it, and that the resource server may give to the client.

4.1. OAuth Error Responses

When a resource server or client attempts to access one of the authorization server endpoints Section 1.5 or a client attempts to access a protected resource at the resource server, 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 request failed authentication, the authorization server or the resource server responds with an OAuth error message as described throughout Section 2 and Section 3.

4.2. UMA Error Responses

When a resource server or client attempts to access one of the authorization server endpoints Section 1.5 or a client attempts to access a protected resource at the resource server, if the request is successfully authenticated by OAuth means, but is invalid for another reason, the authorization server or resource server responds with an UMA error response by adding the following properties to the entity body of the HTTP response:

error REQUIRED. A single error code. Value for this property is defined in the specific authorization server endpoint description.

error_description OPTIONAL. 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 authorization server MUST respond with the HTTP 400 (Bad Request) status code.

For example:

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

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

5. Specification of Additional Profiles

This specification defines a protocol that has optional features in it. For interoperability and deployment purposes to serve particular use cases, third parties may want to define profiles of the UMA core protocol that restrict these options.

Further, this specification has two modular and extensible elements of its design that are specified in terms of specific kinds of profiles:

- RPT token formats and associated sub-protocol flows: These are specified in terms of RPT profiles.

- Claims-gathering sub-protocol flows and specific claim types: These are specified in terms of claim profiles.

Likewise, third parties may want to define additional token and claim profiles to achieve interoperability and deployment success for particular use cases. It is not practical for this specification to standardize all of these additional profiles. However, to serve overall interoperability goals, the following sections provide guidelines for third parties that wish to specify profiles of UMA.

5.1. Specifying Profiles of UMA

It is STRONGLY RECOMMENDED that profiles of UMA document the following information:

1. Specify a URI that uniquely identifies the profile.

2. Identify the responsible author and provide postal or electronic contact information.

3. Supply references to previously defined profiles that the profile updates or obsoletes.

4. Specify relevant authorization server configuration data defined and/or utilized by the profile.

5. Specify the set of interactions between endpoint entites involved in the profile, calling out any restrictions on ordinary UMA-conformant operations and any extension properties used in message formats.

6. Identify the legally responsible parties involved in each interaction and any new obligations imposed, in the fashion of [UMA-obligations].

7. Define any additional or changed error states.

8. Supply any additional security and privacy considerations, including analysis of threats and description of countermeasures.

5.2. Specifying RPT Profiles

It is STRONGLY RECOMMENDED that RPT profiles document the following information:

1. Specify a URI that uniquely identifies the token profile.

2. Identify the responsible author and provide postal or electronic contact information.

3. Supply references to previously defined token profiles that the token profile updates or obsoletes.

4. Specify the keyword to be used in HTTP Authorization headers with tokens conforming to this profile.

5. Specify relevant authorization server configuration data defined and/or utilized by the token profile. At a minimum, specify the keyword for an authorization server to supply in the value of the "uma_token_profiles_supported" property to advertise its support for this token profile.

6. Specify the syntax and semantics of the data that the authorization server associates with the token.

7. Specify how the token data is associated with, contained within, and/or retrieved by means of, the on-the-wire token string.

8. Specify processing rules for token data.

9. Identify any restrictions on grant types to be used with the token profile.

10. Define any additional or changed error states.

11. Supply any additional security and privacy considerations.

12. Specify any obligations specific to the token profile, in the fashion of [UMA-obligations].

See Section 3.3.1 for an example.

5.3. Specifying Claim Profiles

It is STRONGLY RECOMMENDED that claim profiles document the following information:

1. Specify a URI that uniquely identifies the claim profile.

2. Identify the responsible author and provide postal or electronic contact information.

3. Supply references to previously defined claim profiles that the claim profile updates or obsoletes.

4. Specify relevant authorization server configuration data defined and/or utilized by the claim profile. At a minimum, specify the keyword for an authorization server to supply in the value of the

"claim_profiles_supported" property to advertise its support for this claim profile.

5. Specify the syntax and semantics of claim data and requests for claim data.

6. Specify how an authorization server gathers the claims.

7. Define any additional or changed error states.

8. Supply any additional security and privacy considerations.

9. Specify any obligations specific to the claim profile, in the fashion of [UMA-obligations].

See Section 3.5.1.1 for an example.

6. Security Considerations

This specification relies mainly on OAuth security mechanisms for protecting the host registration endpoint at the authorization server 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 protection API token (PAT) 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 PAT, particularly if used in "bearer token" fashion.

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

This specification defines a number of JSON-based data formats. As a subset of the JavaScript scripting language, JSON data SHOULD be consumed through a process that does not dynamically execute it as code, to avoid malicious code execution. One way to achieve this is to use a JavaScript interpreter rather than the built-in JavaScript eval() function.

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

7. Privacy Considerations

The authorization server comes to be in possession of resource set information (such as names and icons) that may reveal information about the user, which the authorization server's trust relationship with the host is assumed to accommodate. However, the client is a less-trusted party (in fact, entirely untrustworthy until it acquires permissions for an RPT in UMA protocol phase 2. This specification recommends obscuring resource set identifiers in order to avoid leaking personally identifiable information to clients 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-obligations].

8. Conformance

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

This specification has dependencies on other specifications, as referenced under the normative references listed in this specification. Its dependencies on some specifications, such as OpenID Connect ([OCStandard] and [OCMessages]), are optional depending on whether the feature in question is used in the implementation.

The authorization server's configuration data provides a machine-readable method for it to indicate certain of the conformance options it has chosen. Several of the configuration data properties allow for indicating extension features. Where this specification does not already require optional features to be documented, it is RECOMMENDED that authorization server developers and deployers document any profiled or extended features explicitly and use configuration data to indicate their usage. See Section 1.5 for information about providing and extending the configuration data.

9. IANA Considerations

This document makes no request of IANA.

10. Acknowledgments

The current editor of this specification is Thomas Hardjono of MIT.

The following people are co-authors:

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

11. Issues

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

12. References

12.1. Normative References

[DynClientReg]
Richer, J., "OAuth Dynamic Client Registration Protocol", November 2012, <https://datatracker.ietf.org/doc/draft-ietf-oauth-dyn-reg/>.

[OAuth-SAML]
Campbell, B., "SAML 2.0 Bearer Assertion Profiles for OAuth 2.0", November 2012, <http://tools.ietf.org/html/draft-ietf-oauth-saml2-bearer>.

[OAuth-bearer]
"The OAuth 2.0 Authorization Framework: Bearer Token Usage", October 2012, <http://tools.ietf.org/html/rfc6750>.

[OAuth-introspection]
Richer, J., "OAuth Token Introspection", November 2012, <http://tools.ietf.org/html/draft-richer-oauth-introspection>.

[OAuth-resource-reg]
Hardjono, T., "OAuth 2.0 Resource Set Registration", December 2012.

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

[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.

[RFC6415] Hammer-Lahav, E., "Web Host Metadata", October 2011, <http://tools.ietf.org/html/rfc6415>.

[UMA-obligations]
Maler, E., "Binding Obligations on UMA Participants", April 2012, <http://kantarainitiative.org/confluence/display/uma/UMA+Trust+Model>.

12.2. Informative References

[OAuth-linktypes]
Mills, W., "Link Type Registrations for OAuth 2", October 2012, <http://tools.ietf.org/html/draft-wmills-oauth-lrdd>.

[OAuth-meta]
Sakimura, N., "JSON Metadata for OAuth Responses", December 2012,

<http://tools.ietf.org/html/draft-sakimura-oauth-meta>.

[UMA-casestudies]
Maler, E., "UMA Case Studies", December 2012, <http://kantarainitiative.org/confluence/display/uma/Case+Studies>.

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

[UMAnitarians]
Maler, E., "UMA Participant Roster", 2012, <http://kantarainitiative.org/confluence/display/uma/Participant+Roster>.

Appendix A. Document History

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

From I-D rev 03 to rev 04, the following major changes have been made:

- The requirement to support the client_credentials flow has been removed.
- The requester access token has been split into two tokens, and all of the tokens have been renamed. The host access token is now the PAT. The requester access token used at the AM's API is now the AAT, and consists of vanilla OAuth. The requester access token used at the host is now the RPT.
- The token and user authorization endpoints for the different APIs at the AM have been joined together, and are now distinguished through the "http://docs.kantarainitiative.org/uma/scopes/prot.json" scope (for the protection API) and the "http://docs.kantarainitiative.org/uma/scopes/authz.json" scope (for the authorization API).
- The token status description format and JSON media type, and the RPT/permission delivery response, have been updated to reflect the RPT naming.
- The configuration data format has changed to reflect the changes above.

- The Phase 2/3 flow has changed and been simplified to match the requirements of the new AAT and RPT.

- Token types are now called token profiles, and this is reflected in the configuration parameter names. Claim types are now called claim profiles, and this is also reflected in the configuration parameter name.

- The requester now asks for permission in a back-channel interaction, and the AM now produces a need_claims error that instructs the requester to use a claims-gathering flow (renamed from "authorization flow").

- Named subsections for token and claim profiles have been added so that they show up in the TOC.

From I-D rev 04 to rev 05, the following major changes have been made:

- The RPT-getting flow and the permission-requesting flow have been separated back out, with two distinct endpoints, RPT and permission request.

- The configuration data format has changed to reflect the changes above.

Author's Address

Thomas Hardjono (editor)
MIT

Email: hardjono@mit.edu