Distributed Authorization

...as conceived by UMA

18 Oct 2012
Eve Maler, UMA WG chair
@UMAWG, @xmlgrrl
UMA turns online sharing with anyone into a “privacy by design” solution

I want to share this stuff selectively!

- Among my own apps
- With family and friends
- With organizations

I want to protect this stuff from being seen by everyone in the world!

Historical
Biographical
Reputation
Vocational
Artistic/user-generated
Social
Location/geolocation
Computational
Genealogical
Biological/health
Legal
...

Authorizing User

Manage
Control

Host
Protected Resource

Authorization Manager

PEP
Protect
PDP

Authorize

Access

Requester

Requesting Party

Delegate
UMA gives users a true digital footprint control console

- Web 2.0 access control is inconsistent and unsophisticated
- To share with others, you have to list them literally
- You have to keep rebuilding your “circles” in new apps
- You can’t advertise content without giving it away
- You can’t get a global view of who accessed what

- You can unify access control under one app
- Sharing policies can test for claims like “over 18”
- You can reuse the same policies with multiple sites
- You can control access to stuff with public URLs
- You can manage and revoke access from one place
UMA leverages OAuth 2.0 and OpenID Connect

OpenID Connect
- You achieve federated single sign-on and login-time attribute exchange
- You control access to claims about you

OAuth 2.0
- You control access to web APIs
- Apps can use a variety of access token types

UMA
- You can grant access to apps operated by anyone
- You control access to a variety of protected resources
- You can grant access by setting policies and terms ahead of time
- The authorization function is standard and centralizable
- Requesting party is authorized based on claims

References normatively as an option (sharing some features as a result)
Thoughts on UMA, vis à vis XACML

- As RESTful, resource-oriented, and web-dev-friendly as possible, and rooted in OAuth by design
- Explicitly enables a “policy self-administrator”
- Enables extreme loose coupling between AM and host
- By default, this separation is “not-quite-PDP” and “slightly-more-than-PEP”
  - AM is also, implicitly, a PAP and PIP
- Policy expression and evaluation are out of band
  - AM integration with XACML policy would be valuable!
Enterprise use cases are coming to the fore

• Use case: organizational API authorization
  – The authorizing party is the enterprise
  – Its agent is a policy administrator
  – It controls what parties access what scopes at what endpoints
  – Akin to traditional enterprise access management, for the “API economy”

• oxAuth (http://ox.gluu.org/jira/browse/OXAUTH) already implements OAuth 2.0 and OpenID Connect
  – Including session management
  – The team is finding it relatively easy to add UMA support
oxAuth sequence diagram

oxAuth Use Case is here
Policies are outside of the UMA core spec but are required for the UMA flow (see 3.5 of spec). In general it is expected to have automatic authorization without human interaction (from oxAuth side). Once policies are satisfied client is authorized. oxAuth allows the oxAuth admin manually authorize client.
UMA defines how to protect three APIs

Host presents an application-specific protected resource API to requester for attempting access.
Requires an UMA requester permission token (RPT)

AM presents UMA protection API to host for registering resources, checking token status, etc.
Requires an OAuth protection API token (PAT)

AM presents UMA authorization API to requester for requesting access, providing claims, etc.
Requires an OAuth authorization API token (AAT)
With a host and AM run by different companies, responsibility matters

All host (auditing only)

Host makes authz decisions; “AM” is just informed of them for auditing reasons

Balanced (“not quite PDP/PEP”)

Host is in charge of resources; AM is in charge of protection; work is divided for privacy, liability, “single hub” reasons

All AM (classic PDP/PEP)

AM knows everything about all resources being protected; host hands over all responsibility

host manages resources; AM protects them
UMA’s “Binding Obligations” spec attempts to account for responsibility.

**R4.** When the AM issues an RPT to a Requester, the Requesting Party using that Requester gains an obligation to the Host Operator to represent the legitimate bearer of the RPT whenever it presents this token to the Host.

Comments: In the case where the "UMA bearer token profile" is being used, the token cannot be bound in any meaningful way to the specific requester and requesting party it applies to, so the Requesting Party takes on the obligation of protecting the RPT from theft and not maliciously sharing the RPT to be used by others. Defining and using different UMA token profiles can mitigate the risk of a failure on the Requesting Party’s part.
The RPT is extensible

<table>
<thead>
<tr>
<th>Authorization data provided by AM</th>
<th>Token format on the wire</th>
<th>Assertion with protected content that the host can locally unpack</th>
<th>Artifact that the host must dereference with the AM at run time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Standardized</strong> as a MTI UMA token profile called “bearer” PDP-- / PEP++</td>
</tr>
<tr>
<td><strong>Permissions</strong> (entitlements with a validity period)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Authorization decision</strong> (XACML-like true PDP / PEP)</td>
<td>Work to define UMA token profile about to get underway</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Claims</strong> (done in many OAuth deployments, proprietarily)</td>
<td></td>
<td></td>
<td>Anticipate interest due to OAuth pattern</td>
</tr>
<tr>
<td><strong>Policies</strong> associated with the requested resource (&quot;sticky policy&quot;-like)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The authorization data associated with a "bearer" token

abstract; meaning is “owned” by host

scopes akin to OAuth’s, but with JSON metadata

HTTP/1.1 200 OK
Content-Type: application/uma-rpt-status+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
  }
]
Next steps for UMA

• Continue to revise the spec (now at rev 05*) in response to experience and comments
  – Including defining additional UMA token profiles
• Conduct interop testing through the OSIS wiki**
• Support implementers and deployers
• Facilitate open source
• Liaise with AXN and other actors in the broader “trusted identities in cyberspace” ecosystem
  – Including the XACMLTC, if there’s interest?
• More webinars and tweet chats…

* http://kantarainitiative.org/confluence/display/uma/UMA+1.0+Core+Protocol
** http://osis.idcommons.net/wiki/UMA1:UMA_Interop_1
Questions?
Thank you

tinyurl.com/umawg | tinyurl.com/umafaq | tinyurl.com/umav1
tinyurl.com/umatrust | tinyurl.com/umaiop | tinyurl.com/umawgfb