Lots to cover so let’s jump in

• A User-Managed Access timeline
• UMA architecture in the OAuth and OpenID Connect context
• UMA use cases
• UMA flows
• Demonstration focusing on an enterprise use case and “interactive claims gathering”
• Walkthrough focusing on a consumer health IoT use case and “pushed claims”
• Q&A
The UMA standard’s progress

- Mar ‘15: UMA V1.0 ratified as Recommendations
- Dec ‘15: UMA V1.0.1 ratified as Recommendations
- Jul ‘17: 1st Public Comment/Review period ends
- Sep ‘17: 2nd Public Comment/Review period ends
- Jan ‘18: Final Recommendations published
- Jan ‘18: Draft UMA Business Model Report published
- Jun ‘18: Business model/IRM cxn jells
- Jan ‘18: Draft UMA Business Model Report published
- May 18: Vendors supporting UMA2: Gluu, ForgeRock, Keycloak (WSO2 coming Q3)
- Feb ‘18: Charter update
- May ‘18: Draft UMA Business Model Report published

Specs refactored, over 100 issues closed, lots of implementation input received, Disposition of Comments doc written…

UMA2 logo (designed by @domcat)
OAuth is for constrained delegation to apps
It has helped to kill the “password anti-pattern”
OAuth is for constrained delegation to apps
It has helped to kill the “password anti-pattern”
OAuth is for constrained delegation to apps
It has helped to kill the “password anti-pattern”
OAuth is for constrained delegation to apps
It has helped to kill the “password anti-pattern”

- Authorization server
- Resource server
- Resource owner
- Client

- Authorizes (consents) at run time after authenticating, at the AS
- App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s
- Standard OAuth endpoints for authorization and access token issuance
OAUTH is for constrained delegation to apps
It has helped to kill the “password anti-pattern”
OAuth is for constrained delegation to apps
It has helped to kill the “password anti-pattern”

Authorization server

Resource server

Resource owner

Client

Authorization Request

Authorization Grant

Authorization Grant

Authorization Grant

Access Token

Access Token

Access Token

Protected Resource

Protected Resource

Some number of API endpoints that deliver the data or other value-add

Standard OAuth endpoints for authorization and access token issuance

App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s

Authorizes (consents) at run time after authenticating, at the AS

App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s

Authorizes (consents) at run time after authenticating, at the AS

Some number of API endpoints that deliver the data or other value-add

Standard OAuth endpoints for authorization and access token issuance

App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s

Authorizes (consents) at run time after authenticating, at the AS

Some number of API endpoints that deliver the data or other value-add

Standard OAuth endpoints for authorization and access token issuance

App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s

Some number of API endpoints that deliver the data or other value-add

App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s

Some number of API endpoints that deliver the data or other value-add
OAuth is for constrained delegation to apps
It has helped to kill the “password anti-pattern”

This can come with a refresh token for renewal without the RO’s intervention

App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s

Standard OAuth endpoints for authorization and access token issuance

Some number of API endpoints that deliver the data or other value-add

Authorizes (consents) at run time after authenticating, at the AS

© 2018 ForgeRock. All rights reserved.
OAuth is for constrained delegation to apps
It has helped to kill the “password anti-pattern”

Authorization Request

Authorization Grant

Authorization Grant

Authorization Grant

Authorization Grant

Access Token

Access Token

Access Token

Access Token

Protected Resource

The RO can revoke the token to withdraw authorization (consent)

Authorized (consents) at run time after authenticating, at the AS

App gets consent based on the API scopes it requested; it has its own identity distinct from the RO’s

Standard OAuth endpoints for authorization and access token issuance

Some number of API endpoints that deliver the data or other value-add

This can come with a refresh token for renewal without the RO’s intervention

(A) Authorization Request

(B) Authorization Grant

(C) Authorization Grant

(D) Authorization Grant

(E) Access Token

(F) Access Token

Resource owner

Client

Authorization server

Resource server
OpenID Connect does modern-day federation
It is an OAuth-protected identity API, plus a bit more
OpenID Connect does modern-day federation
It is an OAuth-protected identity API, plus a bit more

Resource owner

Client

Authorization server

Resource server

= Federation user
OpenID Connect does modern-day federation
It is an OAuth-protected identity API, plus a bit more

Resource owner = Federation user
Client = Relying party
Authorization server
Resource server
OpenID Connect does modern-day federation
It is an OAuth-protected identity API, plus a bit more

- Resource owner = Federation user
- Client = Relying party
- Authorization server = Identity provider ("OpenID provider")
- Resource server
OpenID Connect does modern-day federation
It is an OAuth-protected identity API, plus a bit more

Along with access and refresh token, this endpoint also typically delivers an “ID token” similar to a SAML assertion
OpenID Connect does modern-day federation
It is an OAuth-protected identity API, plus a bit more

Resource owner
= Federation user

Client
= Relying party

Authorization server
= Identity provider
(“OpenID provider”)

Resource server

Along with access and refresh token, this endpoint also typically delivers an “ID token” similar to a SAML assertion

Standard UserInfo endpoint can be called with an access token to look up identity claims
User-Managed Access is for cross-party sharing

UMA brings next-gen delegation and consent to OAuth

- Resource owner
- Client
- Authorization server
- Resource server

tinyurl.com/umawg
@UMAWG
User-Managed Access is for cross-party sharing
UMA brings next-gen delegation and consent to OAuth

Resource owner

Authorization server

Requesting party

Client

tinyurl.com/umawg
@UMAWG
User-Managed Access is for cross-party sharing
UMA brings next-gen delegation and consent to OAuth

Requesting party

Client

Authorization server

Resource server

Resource owner

tinyurl.com/umawg
@UMAWG

kantara

UMA
User-Managed Access is for cross-party sharing
UMA brings next-gen delegation and consent to OAuth

Requesting party

Client

Authorization server

Resource owner

Resource server

Resource server

Resource server

tinyurl.com/umawg
@UMAWG

A authorization
T token
D discovery
R resource registration
P permission
I token introspection
C claims interaction
User-Managed Access is for cross-party sharing

UMA brings next-gen delegation and consent to OAuth

Resource owner

At run time

UX

Opt in

Requesting party

Client

Authorization server

Resource server

Resource server

Resource server

A T

D R P I

C

A authorization
T token
D discovery
R resource registration
P permission
I token introspection
C claims interaction

tinyurl.com/umawg
@UMAWG
User-Managed Access is for cross-party sharing

UMA brings next-gen delegation and consent to OAuth

UX
Share
Opt in

Ahead of time
At run time

Requesting party
Client

Authorization server
Authorization
token
discovery
resource registration
permission
token introspection
claims interaction

tinyurl.com/umawg
@UMAWG

© 2018 ForgeRock. All rights reserved.
User-Managed Access is for cross-party sharing
UMA brings next-gen delegation and consent to OAuth

Requesting party

Client

UX
Share
Opt in
Approve

Authorization server
Authorization
Token
Discovery
Resource Registration
Permission
Token Introspection
Claims Interaction

Ahead of time
At run time
After the fact

tinyurl.com/umawg
@UMAWG
User-Managed Access is for cross-party sharing

UMA brings next-gen delegation and consent to OAuth

UX

Share Monitor Opt in Approve

Ahead of time Anytime At run time After the fact

Requesting party

Client

Authorization server

Authorization

server

A

T

D

R

P

I

C

Resource server

Resource server

Resource server

tinyurl.com/umawg
@UMAWG

25
User-Managed Access is for cross-party sharing
UMA brings next-gen delegation and consent to OAuth

Resource owner

Requesting party

Client

Ahead of time
Anytime
Anytime
At run time
After the fact

UX
Share
Monitor
Withdraw
Opt in
Approve

Authorization server

Resource server

Resource server

Resource server

A
T
Authorization
server

D
R
P
I

C

A
T
authorization
token
discovery
resource registration
permission
token introspection
claims interaction

tinyurl.com/umawg
@UMAWG
Like OpenID Connect for *identity*, UMA adds an API *access management* layer to OAuth2

Some use cases for UMA:

- Enterprise API protection
- For financial consumers
  - Discovering and aggregating UK pension accounts and sharing access to financial advisors
- In industrial and consumer IoT
  - For proactively or dynamically sharing smart device control or data with others
- Healthcare
  - As profiled in the Health Relationship Trust (HEART) WG at OpenID Foundation
  - Part of the new OpenMedReady framework for trustworthy remote care
To sum up: UMA enhances OAuth as follows

**The UMA2 Grant spec adds to OAuth2**

- The resource owner authorizes protected resource access to clients used by entities that are in a requesting party role. This enables *party-to-party authorization*, rather than authorization of application access alone.
- The authorization server and resource server interact with the client and requesting party in a way that is *asynchronous* with respect to resource owner interactions.
- This lets a resource owner *configure an authorization server with policy conditions at will*, rather than authorizing access token issuance synchronously just after authenticating.

**The UMA2 Federated Authorization spec adds to the UMA2 Grant**

- *Multiple* resource servers operating in different domains can communicate with a *single* authorization server operating in yet another domain that acts on behalf of a resource owner.
- A service ecosystem can thus automate resource protection, and the *resource owner can monitor and control* authorization grant rules through the authorization server over time.
- Authorization grants can *increase and decrease* at the level of individual resources and scopes.
The UMA2 grant of OAuth: the basics

urn:ietf:params:oauth:grant-type:uma-ticket

(see also tinyurl.com/uma2grantwsd)
Other things to note about the UMA2 grant

• Types of token endpoint errors beyond vanilla OAuth:
  • need_info (403) with optional hints about what claims are needed
  • request_submitted (403) for RO action with optional polling interval
  • request_denied (403)

• The AS can issue a persisted claims token (PCT) with an RPT
  • The client can supply the PCT at the token endpoint later, refresh token-like, in hopes it will hasten RPT issuance without RqP involvement

• The client can ask for an RPT to be upgraded
• The client can ask for an RPT to be revoked
• Like some other grants, this one accommodates both public and confidential clients
Breaking apart the authorization server and resource server (externalizing authorization)
(see also tinyurl.com/uma2fawsd)

Protection API endpoints:

- **Resource registration**: Puts resources under AS protection; AS responds with resource IDs; resources can have *unique scopes*

- **Permission**: Requests a *permission ticket* to deliver to the client after the tokenless resource request

- **Token introspection**: Customizes OAuth Token Introspection (RFC 7662) to *enhance* the token introspection response object
Demonstration by Mike
Token has correct scopes?
Token has correct scopes? YES
401/Unauthorized as_uri=".." ticket="..."
Restrict (URL & METHOD) to UMA scopes
SCOPE Name

**Id:** Internal_Partner

**Display Name:** Policies to control access to partner API's.

Authorization Policy:
- Partner_Client
- Business_Hours
- Not_a_Bot

Command: Add Authorization Policy
Sample RPT policy

def authorize(self, context):
    print "RPT Policy. Authorizing ..."

    if context.getClaim("country") == 'US':
        print "Authorized successfully!"
        return True

    return False
Demo Code

https://gluu.co/gg-demo
1. Client calls API with no RPT token

Kong returns as_uri, permission ticket
2. Client obtains oxd token

Needed to call protected oxd endpoints

**Request url:**
https://demo.gluu.org:8443/get-client-token

**Request headers:**

<table>
<thead>
<tr>
<th>Content-Length</th>
<th>Accept-Encoding</th>
<th>Accept</th>
<th>User-Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td>260</td>
<td>gzip, deflate</td>
<td><em>/</em></td>
<td>python-requests/2.5.2 CPython/2.7.6 Linux/3.13.0-149-generic</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>'Connection': 'keep-alive', 'Content-Type': 'application/json'</td>
</tr>
</tbody>
</table>

**Request body:**

```json
{
    "client_secret": "e56c7000-1c66-4db6-b0ef-236f6d243bac",
    "oxd_id": "ae42f6d9-91d8-48d3-8a78-9fd4e29d3ce1",
    "scope": [
        "uma_protection",
        "openid"
    ],
    "client_id": "@!7A1F.7A69.7E9A.EFBA10001!AD32.2532!0008!A073.4849.C31B.861A",
    "op_host": "https://demo.gluu.org"
}
```

**Response status:**

200

**Response headers:**

<table>
<thead>
<tr>
<th>date</th>
<th>content-length</th>
<th>content-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fri, 22 Jun 2018 00:00:15 GMT</td>
<td>148</td>
<td>'application/json'</td>
</tr>
</tbody>
</table>

```json
{
    "status": "ok",
    "data": {
        "access_token": "55bbd556-3909-426b-8028-9f7ad3de049f",
        "scope": "openid uma_protection",
        "expires_in": 299
    }
}
```
3. Client calls `/uma_token` to get RPT

Request url:
https://demo.gluu.org:8443/uma-rp-get-rpt

Request headers:
```
{
    "Content-Length": "157",
    "Accept-Encoding": "gzip, deflate",
    "Accept": "/*",
    "User-Agent": 
CPython/2.7.6 Linux/3.13.0-149-generic",
    "Connection": "keep-alive",
    "Content-Type": "application/json",
    "Authorization": "Bearer 55bd556-3909-426b-8028-9f7ad3de049f"
}
```

Request body:
```
{
    "scope": [
        "demo_scope_non_gathering",
        "uma_protection"
    ],
    "ticket": "f1203ab2-19f4-4407-9db4-f54249e3d87a",
    "oxd_id": "ae42f6d9-91d8-48d3-8a78-9fd4e29d3ce1"
}
```

Response status: 200
Response headers:
```
{
    "date": "Fri, 22 Jun 2018 00:00:15 GMT",
    "content-length": "241",
    "content-type": "application/json"
}
```

Response body:
```
{
    "status": "ok",
    "data": {
        "access_token": "04dca3ea-ae34-4d9-95f0-90e1a6ad6a3c_BE23.D2D9.B87D.C5D9.8F1A.15A6.7C6E",
        "token_type": "Bearer",
        "updated": "false",
        "pct": "91f1518c-633f-4ab0-8750-b68d7c6e2a_B156.673C.210F.319F.6491.C01A.2A8C.FC00"
    }
}
```
4. Client calls API Gateway with RPT

Gluu Gateway returns permission ticket, as_uri

HTTP Request:

```
http://demo.gluu.org:8000/posts
```

HTTP Headers:

```
Accept-Encoding: gzip, deflate
Connection: keep-alive
Accept: */*
User-Agent: python-requests/2.5.2 CPython/2.7.6 Linux/3.13.0-149-generic
Host: non-gathering.example.com
Authorization: Bearer a600cb8d-0c1e-4a8e-b43f-903984c1b66b_9EEC.0E57.C489.551C.1011.34EB.FE73.610E
```

HTTP Status Code: 200

HTTP Response Headers:

```
['Content-Type': 'application/json; charset=utf-8', 'X-Kong-User-ID': '1', 'Content-Length': '0']
```

HTTP Response Body:

```
[
  {
    "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et
    "userId": 1,
  }
]
```

Repository:

```
443 = oxAuth
8000 = kong
8443 = oxd
8080 = client demo
```

HOORAY!

CONTENT
Claims gathering

What if one or more of the policies evaluate to False?
No RPT for you! Go directly to Claims Gathering!

Go to /uma_authz


New

status: "error",

data: {
"error_description": "The authorization server needs additional information in order to determine whether the client is authorized to request the resource",
"details": {
"ticket": "2f2b0c5a-84fe-4fe5-9079-795b1829a5de",
},
"required_claims": [
"claim_type": "string",
"friendly_name": "country",
"name": "country",
"claim_token_format": [
"http://openid.net/specs/openid-connect-core-1_0.html#IDToken"
],
"issuer": {
"https://demo.gluu.org"
}],
"error": "need_info",
"error": "need_info"
}
LIVE DEMO! (ish)

Requesting party is redirected to the AS for a multi-step consent workflow.
Claims gathering done! Here’s a PCT for next time!
Walkthrough by Eve:

Sharing pulse oximeter data in a trusted and consented way with third parties through loosely coupled cloud services.
Dr. Lopez prescribes a pulse oximeter to Lynda Wallace; an administrator provisions it electronically.
When Lynda first logs in to the ACME Medical patient portal, her device is inactive.
After she clicks on the red light, she is asked to consent to device activation and data reading by Dr. Lopez.
After she consents, her device now shows as active, meaning a policy is lodged to allow data sharing and her smartphone is prepared to be a hub.
After pairing the oximeter device to her phone, she logs in to her ACME Medical mobile app using the same identity credentials as on the portal.
The mobile app securely mediates an oximeter data reading, and shows that the reading was successful.
Dr. Lopez logs in to the ACME Medical portal
Dr. Lopez’s view once authenticated is this home screen
In his My Patients view, Dr. Lopez sees a listing with Lynda Wallace and others.
Dr. Lopez selects Lynda Wallace as the patient whose data he wants to view.
He chooses Lynda’s device profile
Because of the policy she consented to activate, Dr. Lopez is able to proceed to view her data.
The User-Managed Access (UMA) 2.0 grant of OAuth:

a) gives his client app a permission ticket on first resource attempt
b) requires an ID token for proof
c) issues an access token
d) requires it for data access
Thank you!
Questions?

Eve Maler | ForgeRock | @xmlgrrl
Mike Schwartz | Gluu | @gluufederation
27 June 2018