Delegated and Chained Authorization with OAuth2 and UMA

Javed Shah
ForgeRock, javed.shah@forgerock.com,+1925679570, 655 Glen Mady Way, Folsom, CA 95630, USA

ABSTRACT

This paper presents an authorization framework for solving chained authorization and delegated authorization problems in HTTP/JSON based distributed authorization architectures. This new framework combines the use of OAuth2 token exchange and the UMA 2.0 Grant flow to enable authorized permission-cascading for end users within the bounds of least privilege. At the end also outlined is future work that needs to be undertaken to improve the effectiveness and adoption of the framework.

CCS CONCEPTS

• **Security and privacy** → Authentication, Authorization

KEYWORDS

access control, authorization, token exchange

1 INTRODUCTION

Companies leading digital transformation projects often have a need to setup authorization policies that are used to enforce access decisions in APIs called on behalf of the end user. Such a system of access enablement requires a complex model to support the creation and management of permissions for protected resources, scopes associated with the resources and various applicable resource-use constraints. It thus becomes necessary to decouple permissions from resource objects and users, and dynamically apply constraints to the scoped use of those resources to authorized users, or agents acting on behalf of those users. A resource's scope is a bounded context of access that is possible to be performed on it, in a sense it is a verb that could be applied to the resource. Besides the need to model the system of users, permissions, roles, resources and constraints, there is also a need to model the dependency of applications on one another, including but not limited to the concept of nested permissions that allows cascading grants to end users based on the initial role assignment. This is termed chained authorization in this paper.

Nowadays, it is considered mission critical to facilitate user consent acquisition and propagation in downstream access decisions. Delegated authorization in this paper refers to the act of letting interim clients holding the end user's authorization token become agents of the end user and act on her behalf when requesting access to downstream resources.

2 AUTHORIZATION FRAMEWORK

Administrators need to be able to organize application permissions and resource-use constraints into manageable profiles that can be grouped together as roles according to business requirements. To facilitate management of large number of such profiles, permissions, resource objects and constraints it is a best practice to allow applications to dynamically register new resources, with the Authorization server. Application (or resource) owners must also be able to create and manage permissions and constraints that affect the registered set of resources.

2.1 User Managed Access 2.0

UMA 2.0 [1] is a generalized framework designed as an extension to OAuth 2.0 [2] allowing resource owners such fine-grained control over protected resources, accessed by clients used 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 policies. A generalized framework for enabling distributed authorization UMA 2.0 is presented in Figure 1.

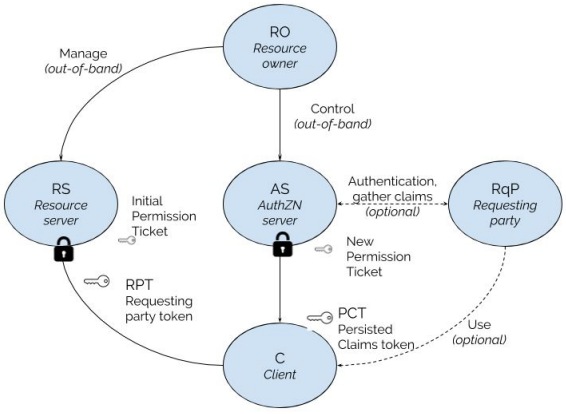

Figure 1: UMA 2.0 Authorization Overview

The UMA 2.0 process largely involves the UMA 2.0 Grant flow, in which a requesting party obtains a RPT to access

the resource, and resource registration which can occur at various stages through the UMA process by the resource owner. The sequence diagram in Figure 2 outlines a successful registration of a protected resource followed by a request for said resource. According to the UMA 2.0 specification, and as also indicated in Figure 1, the RO authorizes protected resource access to clients used by entities that are in a RqP role. This enables party-to-party authorization which is more powerful than the authorization of application access alone. While it is more powerful, it is also more complicated and introduces the notion of Permission Ticket, which is a correlation handle binding requested permissions and passed all around- initially between RS and Client, presented by Client at the AS's token endpoint and during RqP redirects.

2.2 OAuth2 Token Exchange

There is overhead in this token-heavy architecture but is often a necessary evil to ensure secure consent-management and resource sharing between parties. The authorization server and resource server interact with the Client and RqP asynchronously without the RO involved. This lets the RO configure policies at the AS at will, rather than authorizing access token issuance synchronously just after authenticating, which is the traditional OAuth2 authorization grant flow. While this paper does not claim to resolve the security implications of new UMA tokens flowing over the wire, it does attempt to present chained and delegation authorization frameworks that do not rely on a second-level UMA interaction, but instead fall back on a relatively young OAuth2 draft specification: Token Exchange [3].

The OAuth2 Token Exchange draft specification improves upon the conventional OAuth2 flow of exchanging Resource Owner's authorization for an access token by adding a framework for security token exchange. It is important to discuss the semantics of impersonation here. When Requesting Party (client) RqP-A impersonates Trader B, RqP-A is given all the permissions that Trader B has within the scope of that authorization request, and is therefore indistinguishable from Trader B in that context. Thus, when RqP-A impersonates Trader B, then in so far as any entity receiving such a token is concerned, they are actually dealing with Trader B. When RqP-A is impersonating Trader B, RqP-A is Trader B.

2 PERMISSION DESIGN

Here is presented, the design of a permission model that is opaque to the client and decouples resources, constraints, policies and scopes allowing the application owner to create a fine-grained authorization model. Another design goal is to allow chained authorization whererin grant of access to service A automatically grants access to service B. In other words, grant of permission X on Resource A also grants permission Y on Resource B provided certain conditions are met. In the traditional model of RBAC, policy configurators would attach roles to users and enforce coarse-grained access to resources based on those implicit associations.

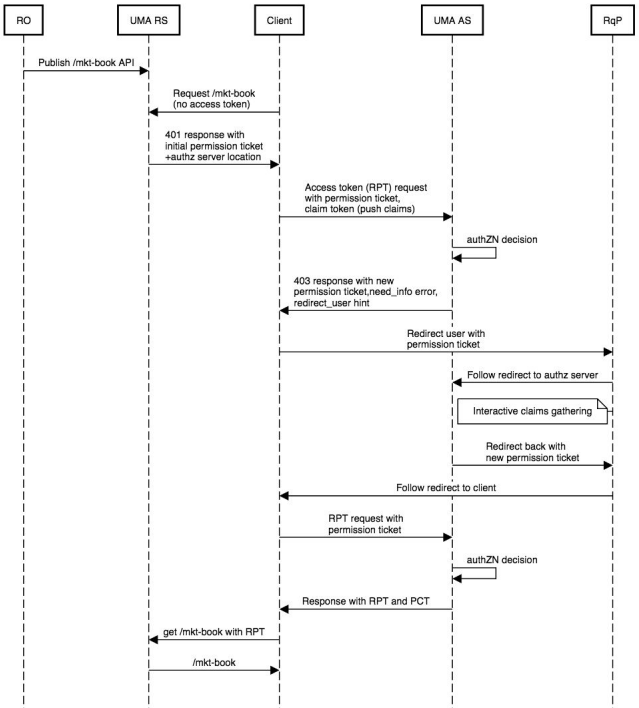

Figure 2: UMA 2.0 Grant Flow

In the distribution authorization model, finer-grained access control can be achieved with loose coupling between the accesses available on a resource, such as scopes, and the actual business logic of establishing a given user's permission to use those scopes in real time. This deliberate decoupling of the 'permission object' from the policies used for enforcing access to the resource makes nested permission design possible. I shall present a few examples for illustrative purposes.

A permission is of the form 'A can do B on resource C', where A represents one or more actors- users, roles and groups, or a combination thereof. B represents the verb- an action to be performed, and C represents the protected resource. For example, we define a top-level permission, namely BOOK-DEAL-PERM: *'TRADER can do VIEW on resource MARKET-BOOK'*. A nested permission can be of the form '**X can do Y on resource Z because X has permission P**'. As an example, we define a nested

permission, namely MARKET-DATA-DEP-PERM: *'TRADER can do VIEW on resource SGX-DELAYED-FUT because TRADER has permission BOOK-DEAL-PERM'*. This translates to a TRADER having VIEW access on delayed SGX Futures data because the TRADER also has VIEW access to the MARKET-BOOK. Permissions can also add resource-use constraints as shown in Table 1 below but most significantly allow a cascade of access as denoted below using the RELAY control.

Table 1: Permission Structure for Dependency

#	Resource	Constraint	Scopes	Dependency
1	/mkt-book	MAX:$2M	VIEW	RELAY #2
2	/sgx-del-fut	6 months	VIEW	-

The permission dependency structure with liaison to policies and decision strategy resolution as shown in Figure 3, allows the cascade of accesses to the holder as deemed appropriate at runtime by policy.

Next is presented a model for indicating delegation semantics inside permissions. Normally, a permission binds resources and scopes to policies for evaluating if the access should be granted and if so, subservient to which constraints. However, there are cases where the evaluated permission grant is positive, or allowed, but only when using a delegate to access the resource. The nature of these use cases will become clear in the section on Delegated Authorization.

Table 2: Permission Structure for Delegation

#	Resource	Constraint	Scopes	Delegation
1	/mkt-book	MAX:$2M	VIEW	-
2	/sgx-del-fut	6 months	VIEW	dtnadmin

A decision strategy is used to resolve conflicting access decisions from the aggregated policies- this could occur both within a permission or within a role as a sum of the outcomes of all permission evaluations. Possible decision strategies are:

- UNION: the number of ALLOW decisions must be greater in number than the number of DENY decisions
- AND: even a single DENY decision will deny access
- OR: even a single ALLOW decision will permit access.

A role represents all possible permissions that should be evaluated to grant access to various resources. If any of the permissions have nested permissions then those also will be evaluated.

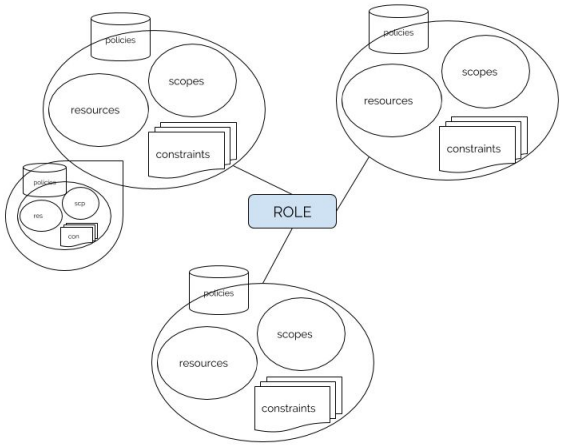

Figure 3: Roles as aggregated top-level permissions

3 CHAINED AUTHORIZATION

Chaining the authorization based on least privilege is a common use case that is sometimes attempted with lesser models such as adding all user claims in the access token. Over time, adding attribute-based claims results in bloated JWTs, higher cost of performing authorization decisions and violates the principle of least privilege.

Chaining the permission grant can be achieved at the time of authorization by adding the RELAY instruction in the parent permission. If the permissions granted to Trader B contain a RELAY to another nested permission then those also will be added to the Requesting Party Token but only conditionally. The condition being that a request must have been made to the protected resource protected by the nested permission, in keeping with the principle of least privilege. The sequence diagram in Figure 4 describes the flow. For simplicity we pick up where Figure 2 left off, ie., when the /mkt-book data has been successfully delivered to the client.

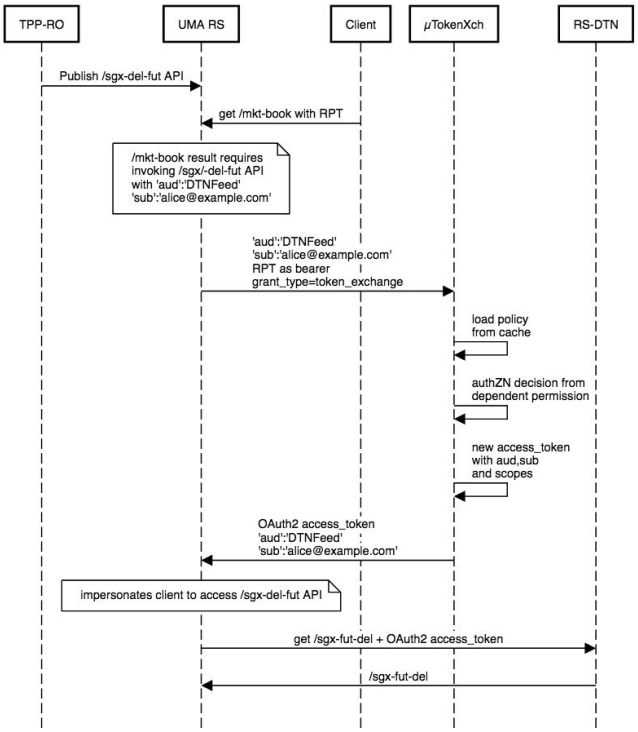

Figure 4: Chained authorization with OAuth2 Token Exchange

There is a new actor at play in the flow: a microservice that implements draft 13 of the OAuth2 Token Exchange specification. The UMA RS acts as a client during the token exchange in order to trade the RPT with a new token that is appropriate to include in a call to the /sgx-del-fut API. The new token is an access token that is more narrowly scoped for the downstream SGX Delayed Futures API and only contains the nested permissions necessary to invoke the API. The Token Exchange microservices is also able to encrypt the granted permissions, as a set of scopes on the resource, per resource server. This provides additional security-by-configuration for the newly minted access token after the token exchange transaction is completed. It should be noted that there is no redirect or direct HTTP request to the UMA AS for obtaining an authorization decision, which is standard procedure for UMA 2.0 grant flow. The proposed model allows for the authorization evaluation to be handled locally at the µTokenXch microservice. There is no redirect or HTTP request needed to the AS even if there is more than one AS in the scenario, because of the idea of caching the nested permissions and policies at the token exchange microservice. Any arbitrary AS should be able to enable notification to the token exchange microservice on policy changes.

4 DELEGATED AUTHORIZATION

In the previous section, Alice the trader had a subscription to DTNFeed, the application that provided a delayed feed of the SGX Futures. Therefore it was deemed okay for the UMA RS to impersonate Alice. This was done by exchanging the RPT for a new access token which represents an authorization grant from Alice's nested permission evaluation by the Token Exchange microservice.

But there are also use cases where the downstream API requires a service identity as current actor, but also must track user identity for audit purposes, as an example. Delegation is expressed in an access token by including the 'sub' and other claims about the primary subject of the access token as well as the actor to whom that subject has delegated some of its rights in a special 'act' claim.

Detection of credit card fraud by a Helpdesk or other system operation follows this paradigm. The operator must immediately cancel the transaction using a Third Party API that requires a service-identity, a service-level privilege, to invoke but also requires the credit card holder's principal information and claims to validate the request. In the context of trading systems, the Trade Desk Manager must assign report execution rights selectively to traders in the business unit. While the Report API requires a service level identity, it also requires claims for the authenticated principal to be passed in to validate depth and extent of runnable and viewable reports. Further, it is quite likely that a chain of delegation actors be required to be built as service invocations result in service A calling service B, service C, and so on, with each service requiring proof of all prior chained authorizations.

The OAuth2 draft token exchange provides delegation semantics and a framework for expressing delegation in JWT tokens. The delegated authorization framework presented in this section requires decoration of the RPT access token issued during the UMA 2.0 Grant flow described earlier with a special "may_act" JSON object claim. This claim-set contains a 'sub' claim identifying the party that is being asserted as being eligible to act for the party identified by the JWT containing the "may_act" claim.

The combination of the two claims "iss" and "sub" are sometimes necessary to uniquely identify an authorized actor, while the "email" claim might be used to provide additional useful information about that party. An example is presented that illustrates the "may_act" claim within a JWT Claims Set. The claims of the token itself are about alice@broker.com while the "may_act" claim indicates that dtnadmin@dtnfeed.com is authorized to act on behalf of alice@broker.com. An example is shown:

```
{
  "aud":"https://dtnasia.dtnfeed.com",
  "iss":"https://issuer.broker.com",
  "exp":1443904177,
  "nbf":1443904077,
  "sub":"alice@broker.com",
  "may_act":
  {
    "sub":"dtnadmin@dtnfeed.com"
  }
}
```


The decoration of the RPT is relatively easy to do within the AS using conditional claims scripting and most OAuth2 Authorization Servers do provide this capability. The only additional requirement is that the UMA RS must be in possession of the actor's access token, either an OAuth2 token or an RPT obtained using the UMA 2.0 grant flow. The sequence diagram in Figure 5 illustrates the delegated authorization flow using UMA 2.0 and OAuth2 Token Exchange. There is again, no redirect or HTTP request necessary from the UMA RS to the UMA AS, or the token exchange microservice.

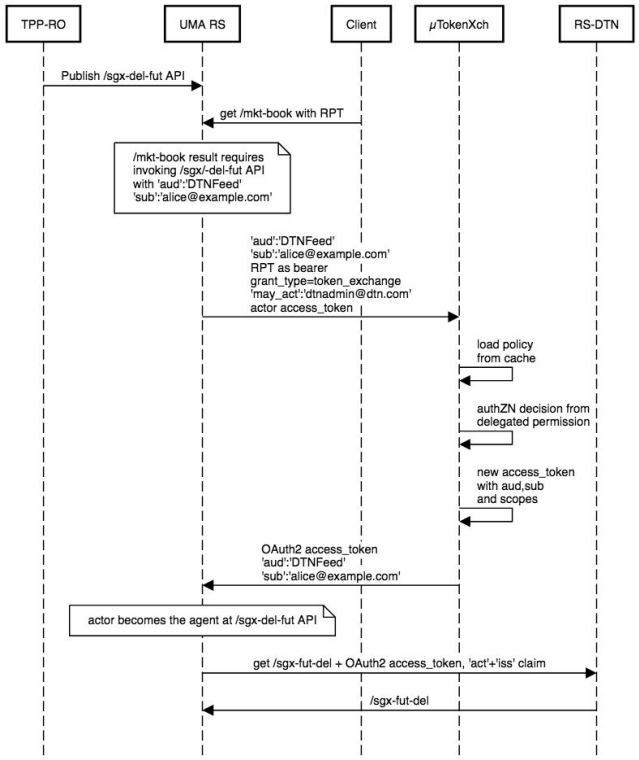

Figure 5: Delegated authorization with OAuth2 Token Exchange

5 SECURITY CONSIDERATIONS

5.1 UMA permission ticket

UMA 1.0 introduced the permission ticket, PT, and UMA 2.0 retains it. The PT introduces a security vulnerability in that when a client redirects an RqP to the claims interaction endpoint on the UMA AS highlighted below in Figure 6, the client provides no discernible context to the UMA AS about which user is appearing at the endpoint, other than implicitly through the permission ticket. A malicious client can therefore impersonate the end-user after the redirect completes and before it returns to the token endpoint at the AS to seek permissions.

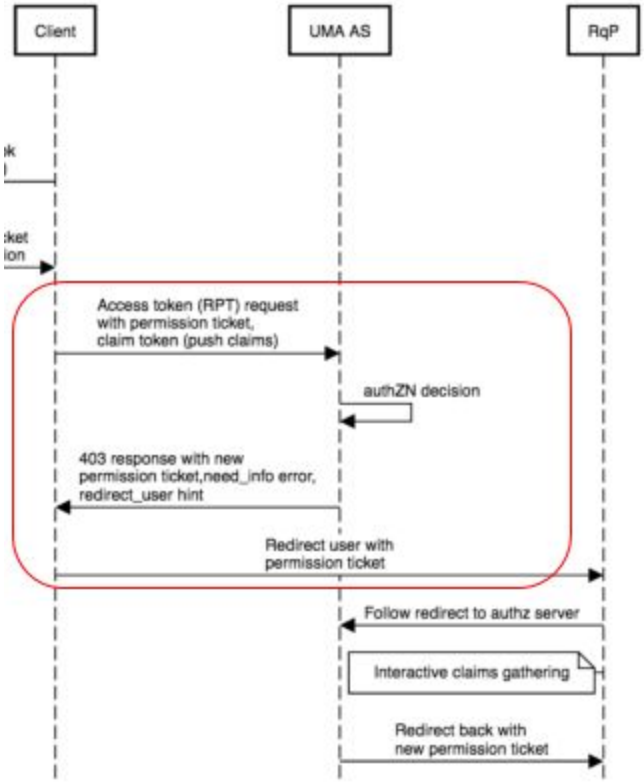

Figure 6: Snippet from the UMA 2.0 Grant flow

5.2 claim_token

The chained and delegated authorization flows presented here do not use the claim_token parameter in the UMA 2.0 grant flow which is susceptible to overloading with claims, especially when untrusted or fraudulent clients attempt to satisfy policies using claim tokens.

6 PERFORMANCE CONSIDERATIONS

The framework presented here requires a notification mechanism from the AS to the Token Exchange microservice to refresh the nested permission sets applicable to the resources the Token Exchange microservices is charged with protecting. In this way any CRUD activities on the nested permissions stored in the AS can be synchronized dynamically to downstream Token Exchange microservices. This is an improvement to existing JSON and HTTP based distributed authorization frameworks that require redirects to the AS for all chained authorization decisions. One way industry practitioners solve this is by adding all known claims about the end user to the JWT, which compromises security by violating the least privilege principle. One other undesirable way this use case is solved in the industry is by using custom business logic built into the RS to detect "flagged" APIs that require additional "permission handling" voiding the decoupling principle necessary for a scalable distributed architecture. In the system described here, when using OAuth2 token exchange with cached nested permissions- or policies attached to those permissions- no calls to the Authorization Server are necessary.

7 FUTURE WORK

Planned work involves framework definition, experiential data collection and results publication. The work is structured into three phases, which are described below. For every phase, the associated research premise, methodology, and expected results are presented, where applicable. The first phase, **P1**, is completed and was concerned with problem definition, use case development, and framework definition. The second phase, **P2**, involves the experimental study using real world data of the effectiveness of the suggested framework. The guiding questions for this phase are: *How can the proposed framework be used with new permission and entitlement models used at ongoing digital transformation projects? What are the performance characteristics in the presence of cloud entities such as authorization servers, resource servers and microservices? What are the performance characteristics of an UMA or OAuth2-only architecture as compared to a mixed UMA or OAuth2 and ABAC architecture?* These questions will be answered by simulating permission and entitlement models collected during field surveys. The third phase and final phase, **P3**, is concerned with analysis and presentation of the simulation results, and the overall evaluation of the framework in digital transformation projects. The guiding questions are: *How can the results be normalized and presented to the practitioners so they may be able to use the framework effectively?* To answer these questions, the simulations will be shared with practitioners, such as security architects and developers. Inputs from these groups will be used to create permission and entitlement examples for documentation aimed at helping increase adoption.

8 CONCLUSION

Current JSON and HTTP based distributed authorization architectures are plagued by least privilege violations. The UMA 2.0 specification addresses party-to-party sharing but introduces chattiness between resource servers and the authorization server. In addition, industry practitioners are solving the identity propagation problem using heavy claim stuffing inside JWTs that affects runtime performance at resource starved APIs and microservices. Decentralized management of permissions is the way to go to avoid managing millions of permissions and 100s of thousands of permission sets at the Authorization Server. Decentralization also prevents the AS from being bombarded with token issuance, token validation and token exchange requests by the participating Resource Servers.

This paper presented an alternative framework for using the existing capabilities of UMA 2.0 and OAuth2 Token Exchange specifications to solve common API to API interactions involves heterogeneous services, each with its own security requirement. I discussed ideas related to caching nested permissions and policies locally at the token exchange microservice to promote faster authorization decision times and eliminate round trips to the Authorization Servers. However, there is further work required to be done in terms of collecting data from experiments, modelling permissions and entitlements, and running field surveys to gauge the overall effectiveness of the framework and make it ready for use by the average digital transformation project.

REFERENCES

[1] Maler, E., Ed., "User-Managed Access (UMA) 2.0 Grant for OAuth 2.0 Authorization", September 4, 2017 https://kantarainitiative.org/confluence/display/uma/Home

[2] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012

[3] M. Jones, A. Nadalin, , B. Campbell, Ed., J. Bradley, "IETF OAuth Token Exchange" Draft 13, OAuth Working Group, April 23, 2018. https://tinyurl.com/y7ke6nja https://www.rfc-editor.org/info/rfc6749

[4] Machulak M., Maler E., Catalano D., Moorsel A., "User-Managed Access to Web Resources", October 2010. In *DIM '10: Proceedings of the 6th ACM workshop on Digital identity management*