ALFRESCO METADATA BASED AUTHORIZATION AND ACCESS CONTROL

Supporting GDPR roles without ACL Complexity

Created by:

Ronny Timmermans – CEO and Managing Director at Xenit
Thijs Lemmens - Senior ECM Engineer at Xenit

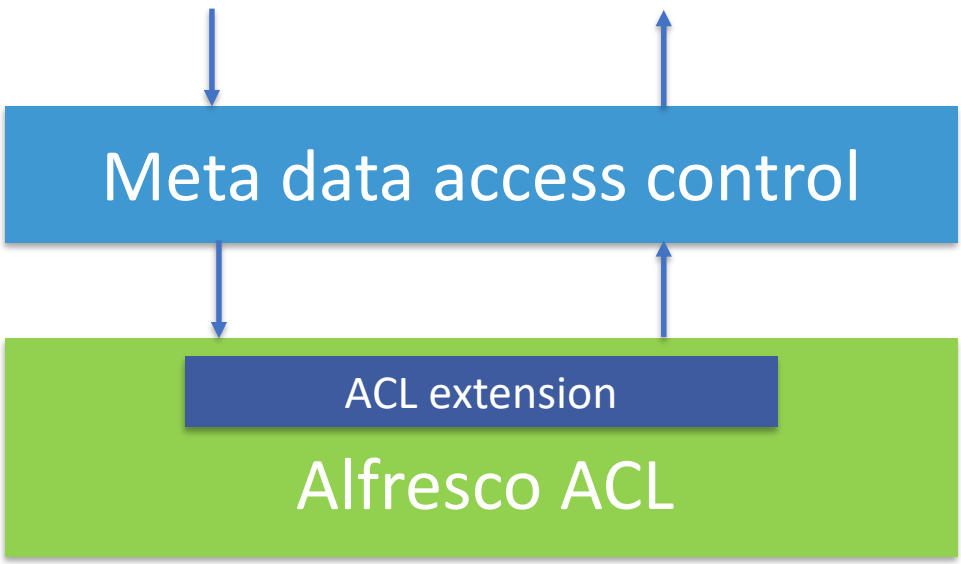
TABLE OF CONTENTS

Introduction

Classical Access Control Lists (ACL) systems, like the one in Alfresco, are powerful for fine grained access control, but cumbersome to maintain and too simple in expressiveness. There is no decent support of AND, OR, NOT type of access rules. Adding an additional protection layer for GDPR would require reviewing your ACL inheritance and creating an additional (oh no, not again) number of Active Directory groups to express privacy constraints. With a dynamic system of user roles and meta-data based authorization rules, you will add a simple (GDPR) filter on top of your current access control policies.

After analysis of the Alfresco permissions model, and our assessment of its fit with GDPR, **we describe how our Meta-data Based Access Control extension empowers Alfresco with flexible meta-data based access control mechanism**. You can combine strategies to best reflect your companies data protection strategy.

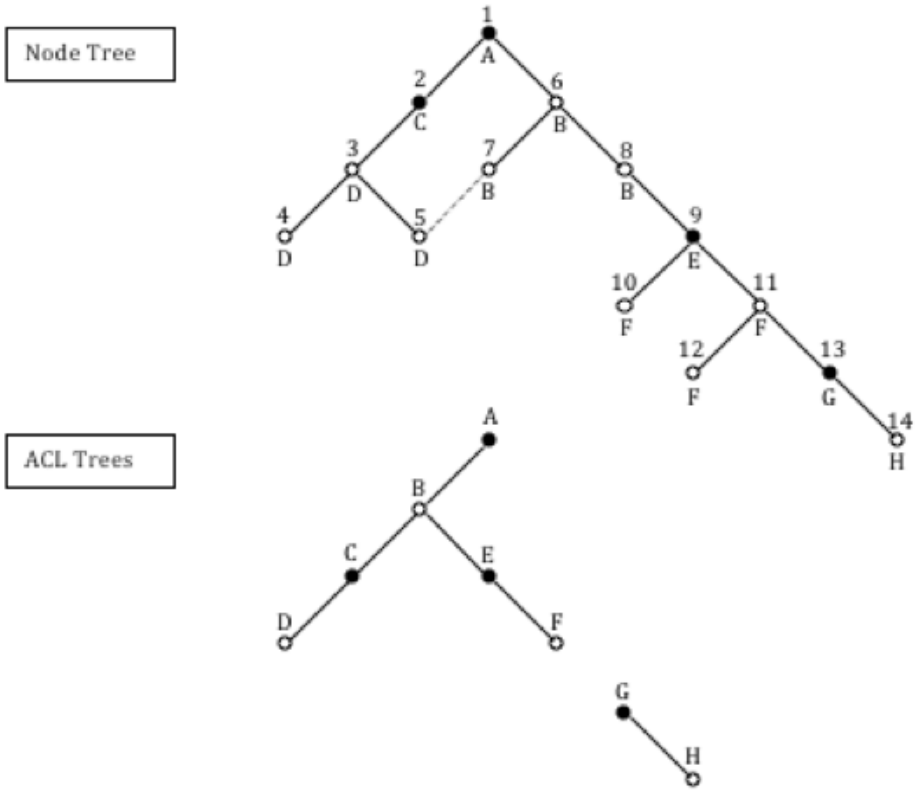
Meta-data and role based access control is simple to maintain, very expressive and works dynamically. You avoid costly re-indexing and you improve scalability and query performance. For a more scientific background and future extensions, read
https://lirias.kuleuven.be/bitstream/123456789/586355/1/paper.pdf



About the permissions paradigm in Alfresco

To ensure the security of documents, Alfresco has put in place a powerful permissions mechanism supported by a generic permission model, making it possible to manage permissions and access control at the finest level.

Permissions in Alfresco (1) are managed through Access Control Lists (ACL). So basically we have alongside the folder structure tree, an (or actually multiple) ACL tree. Every ACL node is associated with one or more nodes from the folder structure (folder/document).



Source : [Alfresco official documentation]

ACL nodes can be linked to parents (parent ACL nodes) in the case they have permission inheritance enabled, or they could be at the very root of their own ACL tree, in the case they have permission inheritance turned off. ACL nodes can have ACL children, inheriting permissions.

(1): The technical details behind permissions in alfresco are a bit more complex than what is presented, we tried to simplify them in this article for the sake of readability and ease of comprehension.

Notes:

- An ACL can either have only one parent or no parents at all, as Alfresco is strictly relying on primary parent for defining inherited permissions, and you can only have one single primary parent for a node.
- Primary parent of a node is by default is the parent referenced when the node was created initially, unless explicitly changed later on.
- ACLs have permission inheritance enabled by default, unless explicitly turned off afterwards.

Every ACL on its own is composed of a set of Access Control Entries (ACE), and every ACE is associated with three different elements, namely:

- **AUTHORITY**: can be a user, group or a role, points to the authority to whom the ACE in question is linked to.

- **PERMISSION**: references the permission/permission-group associated with the ACE

- **ACCESS STATUS**: specifies if the referenced permission is allowed or denied for the associated authority.

When a user does match multiple authorities having different granting ACEs on the same node, that user is granted the most permissive rights amongst them all (2).

(2) Alfresco allows to use deny permissions. DENY is actually not exposed via OOTB Alfresco UI, aka Alfresco Share, and is considered as a low-level interface to be only used on a need-to basis. The default configuration of Alfresco indicates that any deny, for any of the authorities the user is associated to, would deny that user access to that particular permission, even in the case where the user is associated with another authority with an ALLOWED status for the same permission on that same node. That configuration can however be switched off via configuration.

How does that fit with the new GDPR regulations ?

The growing focus around securely managing personal identifiable information (PII) and sensitive information complicates standard ACL based access control.

Any document in your content store can contain PII: a name, an address or an ID number. To enforce additional GDPR protection roles, you have a number of sub-optimal solutions in an ACL based approach.

1. The naive approach would be to classify 1000's or millions of documents in a multi-layered folder structure representing the different criteria for attributing permissions on documents, and then specify permissions on the different layers in that folder structure. The only remaining issue would be to be able to route the documents correctly to the right folder so that they could inherit the right permissions set. Unfortunately, such an approach only works with very simple use-cases, and as soon as the requirements for defining permissions grow complex or change, the approach can hit performance limits or become very hard to maintain. This will lead to an explosion of folders, ACL's and Users groups on top your current operation.

2. Given that access to personal and sensitive information can be detected and mapped to the correct metadata properties, another solution might rely on policies to break permission-inheritance on documents and automatically compute permissions for each document separately. That way, we can be sure that no matter how complex the requirements get, the approach can cover it. However, this leads to a large and uncoordinated sets of policies in need of maintenance and potentially leading to unpredictable side effects. Search performance will be affected with a large number of specific ACL's.

A more intelligent approach is to **add a set of easily configurable filters based upon meta-data access rules.** This approach can be easily combined with a simple folder based ACL system. Obviously, you need to channel all requests to content via an entry gate that enforces the access rules, but that is a good practice anyway. This is a 2 step approach: basic ACL checking, and adding extra filtering based upon meta-data.

Improving Alfresco's ACL system. Simple and scalable.

Alfresco is first and foremost a content repository, and it does a great job at managing content; but its ACL system is not ideal for a number of use-cases :

- Alfresco can batch load nodes (folders/documents) and their properties and aspects, it does not, however, have such facilities for loading ACLs and ACEs.

- While searching for documents, you could load ACLs from the database to assess the access right of the user to the document. This is notably slow for large result sets. Alfresco has externalized read permission-checks to the search component Solr to eliminate this performance bottleneck. But at a cost. Solr needs to track those permissions, which grows into a costly task when you **change access rights** on a folder close to the root of your repository. All changes cascade recursively into all subfolders due to inheritance.

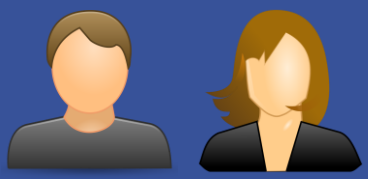
- Wiring GDPR permissions into your whole repository causes a lot stress on the tracking mechanism in Solr and might require almost the same time as a full re-index, possibly causing your search to fall out of sync and operations to be disrupted!

- Every time the policies for attributing access to documents change, all of the previously processed documents would need to be updated. Depending on the volume, this can result in serious load on your system, not only to make the appropriate changes to the ACLs but also for Solr to track them.

How does Alfred Edge and its business API manage (GDPR) access control in Alfresco?

A simple and scalable access control, as imposed by (GDPR) privacy policies, is a cornerstone of our Alfred architecture.

We believe that our Alfred architecture is the key to agility, performance, high availability, privacy, governance and security.



Corporate users

Internet users

Business Application

Customer Portal

ALFRED Edge

API GATEWAY

Logging

Request rewriting

Caching

Security filters

Authorization Providers

Permission Service Extensions

Alfresco

OPERATIONAL

Permission Service Extensions

Alfresco

ARCHIVE

Our reference architecture includes an API Gateway (Alfred Edge) to restrict access to the content repositories and all other applications below the line.. Such gateway gives you more control and more flexibility for every request. Authentication and other overarching building blocks can be centralized and standardized. As an example, Internet users can be authenticated one way (LinkedIn,..), intranet users against an Active Directory.

At the level of the security filters inside our API Gateway we did set up an extension mechanism to decorate the requests with extra attributes in a secure way. One of these decorators looks up security settings in the configuration and sets up the appropriate attributes on the request for expressing Metadata-based grants.

Alfred Edge comes with a role-based metadata grant-provider along with a group-to-role mapper making it extremely easy to configure grants based on business profiles, departments, or customer defined roles.

In very bespoke cases, you can add a custom extension and leverage all available context information (request, authenticated user, groups, roles ….) to set metadata-based grants according to your policies. And of course multiple providers can live side by side without interfering with one another grants, collaboratively setting all assigned grants to the request.

In Alfresco, an extra module is added that uses the information in 2 ways:

- Evaluate access for a single node
- Decorate a search query to return only the results that should be seen by a user.

This approach is a performant way to apply metadata based permissions because the evaluation is done on query time, and no expensive post filtering needs to be done afterwards.

GDPR Example:

Documents in Alfresco can be marked as containing sensitive information using a property "**gdpr:level**". This property can have 3 levels:

- None
- Personal
- Sensitive

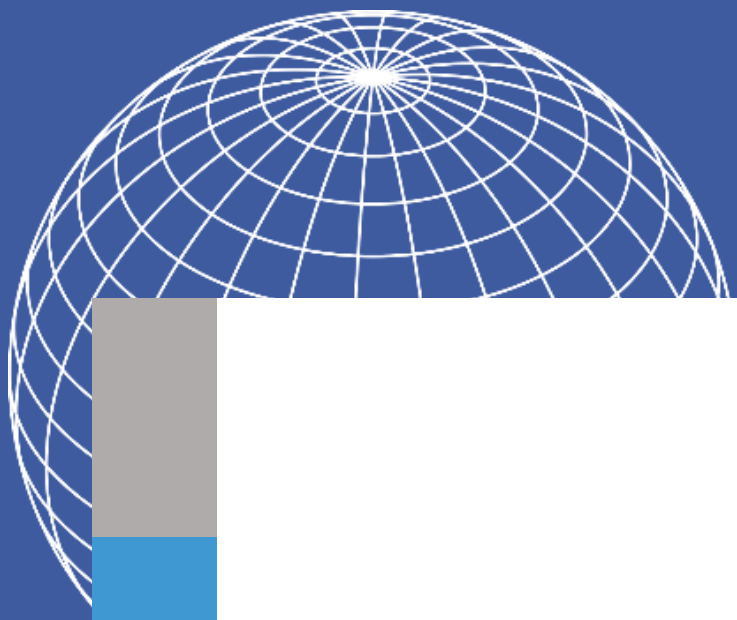
Now, in an LDAP we can have a group for people that can see personal information, and a group for people having access to sensitive information. On every request, Alfred Edge will retrieve this information for the user issuing the request, and pass it over to Alfresco. For a user in the group "personal" Alfresco will then add to the query the constraint that the documents returned should be of *gdpr:level=(None OR Personal)*. Sensitive documents will be excluded in this case.

In this case the configuration is:

```
# Rule applies to property gdpr:level
alfresco-metadata-
permissions.restrictions.properties=gdpr:level
# ordered constraint, lesser or equal
alfresco-metadata-
permissions.restrictions.gdpr\:level.restrictionType=ordered_co
nstraint_leq
# the order of the constraint values, Sensitive Information has
a stronger protection than Personal Data
alfresco-metadata-
permissions.restrictions.gdpr\:level.customConstraintOrder=Pers
onal Data,Sensitive Information
# The header name that should be read from the http request
(Alfred Edge sets the header)
alfresco-metadata-
permissions.restrictions.gdpr\:level.headerName=gdpr_level
```

On the other end, Alfresco is configured to recognize sensitive documents upon the detection of specific types (like a CV), the presence of specific properties (names, ID numbers, ..) and/or even the presence of specific values in specific properties (all configurable through global properties). A common use case is the classification of documents in confidential, internal and public. With a simple grant rule, you define 3 roles and their access to documents carrying such classification.

Upon the reception of a request rooted through the API-Gateway, metadata-based grants are activated, and whenever a permission-check occurs for sensitive content (determined according to our magic-sauce), we consider that assessing the ACL based permissions (Coarse grain permissions) is not enough, and we require some extra specific grants (from within the request attributes) in order to authorize the access !



Conclusion

In summary, using this setup, we actually do not even need to alter any of the permissions set within the repository, we actually keep using them as is. We do however leverage already existing (or newly extracted) metadata and content types to flag content with different levels of sensitivity and then rely on metadata based grants, set at the API Gateway level, to further check permissions on content that is flagged as sensitive. Changing content access policies later on is as simple as changing and reloading configuration with near zero latency.

Our solution fiddles with every query trying to find documents in Solr, in a way that filters out most to all of the un-authorized sensitive documents at the search engine level, and making it way less stressful to do post-query permission filtering on the alfresco side.

Thank you

You can watch a practical demonstration of our metadata based access control, for the GDPR compliance, here:

xenit
return on content