H. Roßnagel, C. H. Schunck, S. Mödersheim, D. Hühnlein (Hrsg): Open Identity Summit 2020,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2020 179

Token Based Authorization

Giovanni Augusto Baruzzi[1]

Abstract: A secure, scalable, fine grained and flexible access control is extremely important for the digital society. The approaches used until now (RBAC, Groups in an LDAP Directory, XACML) alone may not be able to deliver to this challenge. Building from past experiences in the Industry, we propose an Access Management Framework where the central role is played by a token containing all the information needed to implement fine grained access control. This Authorization Token should be signed by the approver and embedded into a "claim" to the application at session time. The application, after checking the validity of the token will control access to the desired resource. In this way we can achieve fine granular access control, scalability and independence from network topologies.

Keywords: Access Control, Token, Fine-grained Access, Authorization, Claim.

1 Evolution of Access Management

1.1 The Beginning

In the early days (1970), to give access to a computing resource, the list of legitimate users was appended to the resource itself. This was then called the Access Control List (ACL).

This first approach had the problem of the ever-increasing administrative effort: each time a resource was added, the administrator had to list all legitimated users again. Similarly, the addition of a new user forced administrators to add her/him to the access list of each resource he may need.

1.2 Role Based Access Management

Role-Based Access Control (RBAC) [FK92] has been introduced to grant access based on the roles that users own in their organization, recognizing that, if a user has a certain role in the organization, he/she must be granted access to a definite, but variable, list of resources. The roles are often associated to user groups which are a list of members: *Users* are assigned to *groups*, and in turn, *groups* associated to *roles* and then *to resources*.

[1] Syntlogo GmbH, Mercedesstraße 1, 71063 Sindelfingen, Germany, giovanni.baruzzi@syntlogo.de

cbe doi:18.18420/provided-by-editor-02

RBAC was a vast improvement compared to the management of simple lists of users for every resource and offered an improved security; the Role Manager could be a different user to the Resource Manager, introducing the first type of segregation of duties.

1.3 The Experience of large-scale Implementations

The experience gathered in large installations (a large German Telecommunication Provider, a large Insurance Company, a major ERP Software Vendor) revealed a number of issues in the implementation of Access Management using pure RBAC: the complexity of modern enterprises prevented the use of RBAC to the organizational roles and the fast life cycle of application, often independently operated, was an obstacle to the original idea to associate all the entitlements needed to a single organizational role.

The adoption of RBAC at application level was very successful, using not organizational roles, but application roles. But even this approach has issues.

1.4 The fine-granular access right

Consider the case where a complex application must manage the access to many resources: the application architect applying RBAC has the possibility to define a very high number of roles. The alternative is to separate the access information into two (or more) pieces: the role itself, defining the function, and additional parameters specifying the resource.

Let's assume that you must define the role of a "Cost Center Chief" but you have a large number of them. Using the pure RBAC, you may be forced to define a long list of roles like "Chief of Cost Center 001", "Chief of Cost Center 002", "Chief of Cost Center 003" and so on.

A more intelligent alternative is to define the functional role "Cost Center Chief" and assign the parameter identifying the cost center. This is what we call fine-granular access management[2].

1.5 Technological Limits of Group Objects

This issue becomes relevant as the numbers of users comes to the millions: the best practice to associate a role with a user group proves as not scalable enough.

[2] The RBAC approach generates an N*M problem (role chief * number of cost centers). With a role parameter this problem is reduced to a more manageable N+M problem

1.6 Reactions in the Industry

Confronted with these problems, the major ERP Software vendor completely defined it's own Access Management (for sure not RBAC) while the Telecomm vendor and the Large Insurance Company USED THE FINE-GRANULAR APPROACH AND added information to the role name, stored as additional user attribute.

2 The Access Management Framework

Building on the experiences cited before, we designed an Access Management Framework built around the idea of storing all access information inside a token and delivering this token at OIDC or SAML session time to the application. The digital signature added to the token allows us to move it around without losing the trust.

2.1 Basic Ideas

1. The ownership of a role is not represented by a membership in a group, but it is stored as a USER ATTRIBUTE, removing the scalability problems.
2. Additional information is added to the role name allowing more granular definitions
3. The format used is JSON and with the addition of a signature it becomes a JSON Web Token (JWT). Libraries to process them are broadly available.

2.2 Authorization Token

The authorization token is a JWT Token containing all the information necessary to define an access right and is the central concept of the proposed Access Management Framework illustrated in Figure 1, below on page 4.[3]

We are going to analyze briefly the entities and the processes that describe this Framework and we are going to see the processes that define, manage and send Authorization Tokens to implement access control for an application. [4]

2.3 The Model

We describe the Framework from different points of view: the interaction view, de-

[3] A word of caution is needed here: we use the JWT formats and the Token is built like a JWT Token but it is never used as Session Token. As a matter of fact, the Authorization Token BUILDS ON OIDC or SAML and is inserted INSIDE a Session Token to be sent as a "Claim" to the application.

[4] The model refers to an SAML or OIDC Environment, where users present to the application a set of *claims*, but is not limited to it.

scribing the processes and the Entity view, describing the responsibilities

2.4 The Interactions

- The “Contract Life Cycle” is the time span between the signature of a contract for an offer from the “Service Organization” and the termination of the contract itself.
- The “Application Life Cycle” marks the time between the start of availability of an application, with the definition of its security model and corresponding role metadata. During the “Application Life Cycle” Authorization Tokens are built, assigned to users or withdrawn from them.
- The “Session life Cycle”. A session starts as a user authenticates to the system and terminates with the logout. During the time span of a session, the system restricts the access to application resources to the legitimated users.

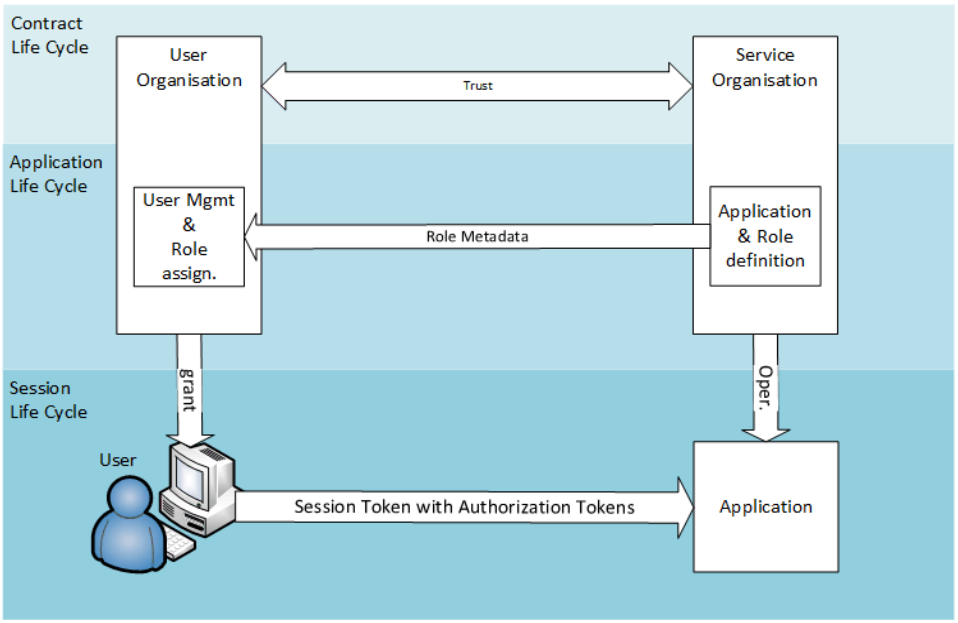

Fig. 1 Authorization Token Framework

3 Life Cycle of an Authorization Token

3.1 Application Architect defines the Roles

The application architect, during the application design process, defines the conditions under which a user can access resources. The resulting set of rules of this process builds the application "Security Model".

3.2 Roles Metadata is being loaded in the IAM (Identity & Access Management System)

After the definition of the application "Security Model", the role metadata must be loaded in the Identity and Access Management. The ID's of Security Domain, Application and Role must be included in the metadata.

3.3 Management of User and assigned Roles

The grant of a role is governed by the typical identity processes and must define beyond the value of the optional parameters a few other information like the granting actor, the assignee and validity dates. As all attributes defined are known, the JWT can be generated and digitally signed by the Customer Organization. This is stored as a user attribute. This signed element is the Authorization Token.

3.4 User accesses an Application

After having authenticated, the user may want to access an application. During this process, he introduces himself to an application presenting a session (access) token containing claims along the SAML [6] or OIDC [7] Standards. One or more of these claims would be Authorization Token.[5]

3.5 Application checks Authorization Tokens and manages access

The Application receives the access token sent in the session, extracts the embedded Authorization Tokens from their "claims", checks their validity, and grants the logged user the corresponding privileges.

[5] As written in note 2: The Authorization Token is never used as Session Token. Instead the Authorization Token is inserted INSIDE a Session Token as a "Claim".

4 Current Deployment

The solution is being used in our IAM System, allowing us to easily implement advanced features like a flexible and easy to understand “Delegated Administration”. Here the role of delegated administrator is coupled with a parameter specifying the set of users for which he is the administrator.

Due to the evolutionary character of this framework and the modest technical effort needed to implement it, the framework has been already successfully used in a financial institution and a government owned agency.

5 Conclusions

The Authorization Token is especially attractive in a Cloud Environment, where the User Organization may be a different one as the Organization providing the application, connected only by the Internet. In such a scenario, it may be very difficult to provide access for the application to the Directories or Databases holding the groups associated with a role or perform provisioning to an Application’s registry. The usage of an Authorization Token solves both problems with a very limited cost.

Bibliography

[FK92] Ferraiolo D.F., Kuhn D.R. : "Role-Based Access Control". 15th National Computer Security Conference: 554–563, 1992.

[CO02] Chadwick D.W., Otenko A., The PERMIS X.509 Role Based Privilege Management Infrastructure ISI, University of Salford, Salford, M5 4WT, 2002.

[WG12] Whitson Gordon: "Understanding OAuth: What Happens When You Log Into a Site with Google, Twitter, or Facebook". https://lifehacker.com/understanding-oauth-what-happens-when-you-log-into-a-s-5918086, 2012

[BS15] Bradley John, Sakimura Nat and Jones Michael: JWT: "JSON Web Token (JWT)". RFC7519, 2015

[EC17] JSON: "The JSON Data Interchange Format", http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, 2017.