Authorization using RBAC

Scenario

Many applications in today's computation scenarios are distributed and composed by services and clients.

Suppose that a certain distributed application uses a remote service in two different servers (assume that the REST architecture (using the HTTP protocol) is used as an implementation) with at least three distinct operations each. Each operation consumes parameters and produces a result. Some of the operations in the first server also call operations on the second server.

In this kind of application we define 3 roles with different privileges to call the services operations. Also we need to define at least three users associated with one or more of the roles. These associations and role privileges can be configured and hold on a file or database, associated with an authentication and authorization server.

We need to enforce the users privileges denying access (not returning a result) to the operations that are not allowed in the roles that the invoking user is on. Also, if the user is allowed to invoke an operation in a service, and that service calls another, that invocation should succeed only if the user is also allowed to invoke directly that second operation.

An administrator should be able to originate the users security data (user identity, roles' definition, allowed operations in each roles, users in each role).

Implementation

A proof of concept for this distributed system should be implemented using some web server software (like, for instance, Node.js). You can define any operations in the servers. On the same computer those servers must run on different ports. The client application(s) should also be web applications supporting any browser. The Authentication/Authorization server (implementing a web service) is a fourth web server in the system.

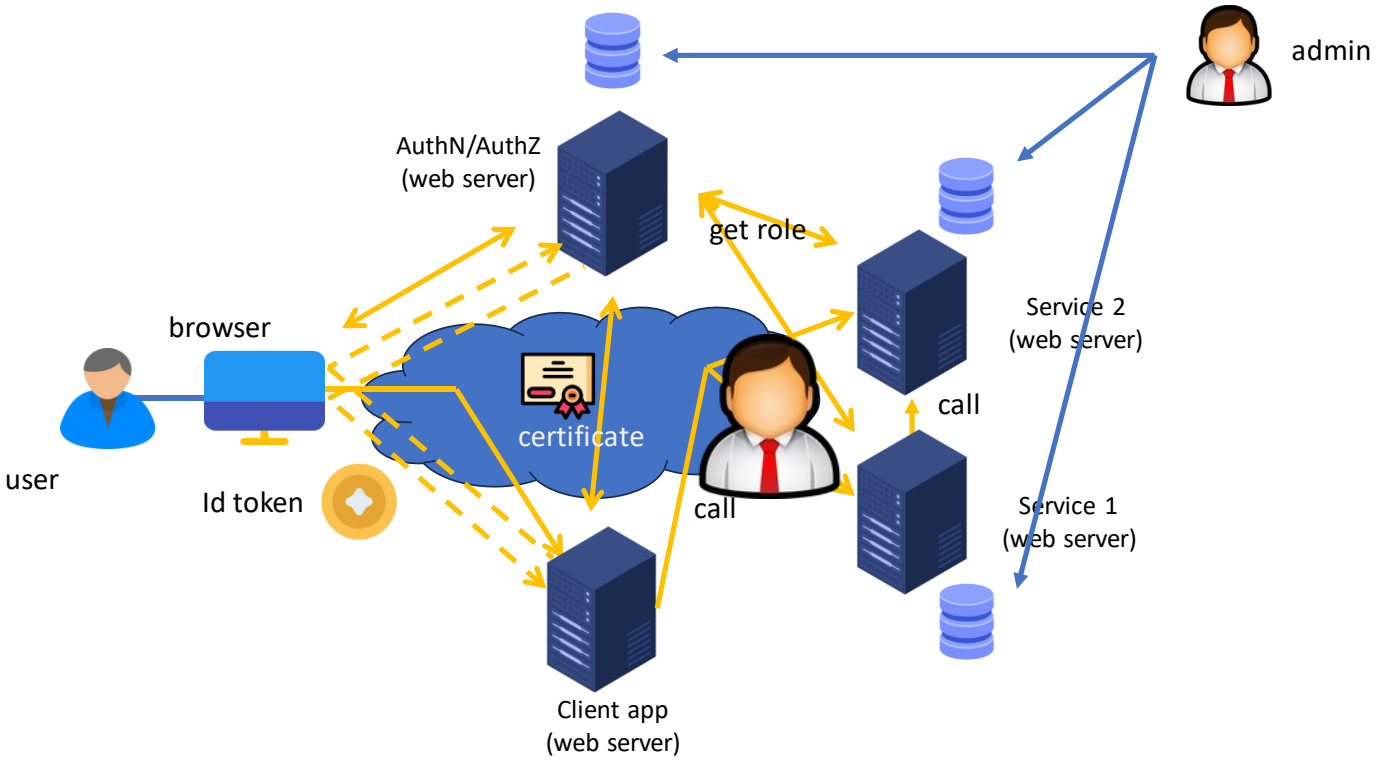
The user authentication mechanism, performed in the Authentication service, should verify the user identity, and when successful, generate a pair of asymmetric cryptographic keys, valid for that user and session with the client application. Also the role of the user should be determined and used in an RBAC (Role Based Access Control) system to allow (or not) the call of operations in the two operation servers (available as links in the client app).

As a suggestion, we can use a simplified version of the OAuth and OpenId protocols for authentication and authorization, as depicted in the following figure.

The private key generated for a user and session can be used to sign **identification** and/or **authorization** tokens (appropriate JSON tokens can be used), sent in a redirection from the

AuthN/AuthZ server, after a successful user authentication. The public key generated is put on a certificate and can be sent (or retrieved) to the client server by its initiative.

All communications should have confidentiality and integrity protection (TLS) and the requests from the client and services servers to the authentication server should also be done in a secure manner. The authentication/authorization data (stored in the AuthN/AuthZ server and/or Service servers) should be specially protected.



You should use trusted libraries in your proof of concept, namely concerning cryptography.

Note that this suggestion is a simplification of standard protocols with proof of possession (PoP).

In a production system the full specifications of OAuth and OpenID, with all security protections, should be used.

Report

The report should describe the architecture that you have chosen for a proof of concept, as well as an explanation of all design decisions that you have made. Consider also the main threats that a system with these functionalities can have, and how they are mitigated in your design and implementation.

Examples and results demonstrating the correct working of the authentication and authorization mechanism should also be included (including the system response to wrong or malicious uses).