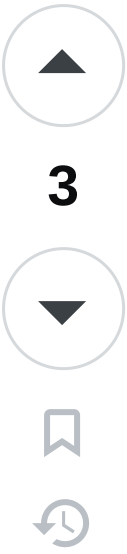
Microservices - how to solve security and user authentication?

Asked 9 years, 3 months ago Modified 9 years, 3 months ago

Viewed 3k times



There is a lot of discussion about microservice architecture. What I am missing - or maybe what I did not yet understand is, how to solve the issue of security and user authentication?

For example: I develop a microservice which provides a Rest Service interface to a workflow engine. The engine is based on JEE and runs on application servers like GlassFish or Wildfly. One of the core concepts of the workflow engine is, that each call is user centric. This means depending of the role and access level of the current user, the workflow engine produces individual results (e.g. a user-centric tasklist or processing an open task which depends on the users role in the process).

In my eyes, thus a service is not accessible from everywhere. For example if someone plans to implement a modern Ajax based JavaScript application which should use the workflow microservice there are two problems:

1) to avoid the cross-scripting problem from JavaScript/Ajax the JavaScript Web application needs to

be deployed under the same domain as the microservice runs

2) if the microservice forces a user authentication (which is the case in my scenario) the application need to provide a transparent authentication mechanism.

The situation becomes more complex if the client need to access more than one user-centric microservices forcing user authentication. I always end up with an architecture where all services and the client application running on the same application server under the same domain.

How can these problems be solved? What is the best practice for such an architecture?

authentication **jakarta-ee** **microservices**

Share

Improve this question

Follow

asked Sep 14, 2015 at 21:18

Ralph

4,818 ● 10 ● 58 ● 96

I am not sure if cross-scripting is the right description of the first issue. What I meant is Cross-Origin Resource Sharing (CORS). – Ralph Sep 14, 2015 at 21:40

2 Answers

Sorted by: Highest score (default)

Short answer: check OAUTH, and manage caches of credentials in each microservice that needs to access other microservices. By "manage" I mean, be careful with security. Specially, mind who can access those credentials and let the network topology be your friend. Create a DMZ layer and other internal layers reflecting the dependency graph of your microservices.

Long answer, keep reading. Your question is a good one because there is no simple silver bullet to do what you need although your problem is quite recurrent.

As with everything related with microservices that I saw so far, nothing is really new. Whenever you need to have a distributed system doing things on behalf of a certain user, you need distributed credentials to enable such solution. This is true since mainframe times. There is no way to violate that.

Auto SSH is, in a sense, such a thing. Perhaps it may sound like a glorified way to describe something simple, but in the end, it enables processes in one machine to use services in another machine.

In the Grid world, the Globus Toolkit, for instance, bases its distributed security using the following:

- X.509 certificates;
- MyProxy - manages a repository of credentials and helps you define a chain of certificate authorities up

to finding the root one, which should be trusted by default;

- An extension of OpenSSH, which is the de facto standard SSH implementation for Linux distributions.

OAUTH is perhaps what you need. It is a way provide authorization with extra restrictions. For instance, imagine that a certain user has read and write permission on a certain service. When you issue an OAUTH authorization you do not necessarily give full user powers to the third party. You may only give read access.

CORS, mentioned in another answer, is useful when the end client (typically a web browser) needs single-sign-on across web sites. But it seems that your problem is closer to a cluster in which you have many microservices that are managed by you. Nevertheless, you can take advantage of solutions developed by the Grid field to ensure security in a cluster distributed across sites (for high availability reasons, for instance).

Complete security is something unattainable. So all this is of no use if credentials are valid forever or if you do not take enough care to keep them secret to whatever received them. For such purpose, I would recommend partitioning your network using layers. Each layer with a different degree of secrecy and exposure to the outside world.

If you do not want the burden to have the required infrastructure to allow for OAUTH, you can either use

basic HTTP or create your own tokens.

When using basic HTTP authentication, the client needs to send credentials on each request, therefore eliminating the need to keep session state on the server side for the purpose of authorization.

If you want to create your own mechanism, then change your login requests such that a token is returned as the response to a successful login. Subsequent requests having the same token will act as the basic HTTP authentication with the advantage that this takes place at the application level (in contrast with the framework or app server level in basic HTTP authentication).

edited Sep 15, 2015 at 20:39

answered Sep 15, 2015 at 19:34

Akira

4,071 ● 2 ● 18 ● 24

Thanks for our answer. I think now I understand that CORS is an minor issue and it looks like that I can solve this easily by extending my JAX-RS service API. The issue 2) looks a little more complex. Because my open source project (Imixs-Workflow) should be open for different use cases. In the past JAAS was my fried on the JEE stack and security was easy to configure and very flexible. But OAUTH looks so complicated to me. I don't want to implement an authentication sever just because I want to allow external clients, using a microservice architecture, get access to my workflow solution. – Ralph Sep 15, 2015 at 20:13

I see. After reading this last message it seems even less that CORS actually solves your problem, since the REST interface in your project is not restricted to Javascript clients in a web browser. I agree that JAAS is awesome and works flawlessly in the Java realm, but when you use open standards, things are different. If you do not want to use OAUTH, then another solution is to implement your own mechanism to generate tokens. I will update my answer accordingly. – Akira Sep 15, 2015 at 20:31

I read more about Token Based Authentication and now I'm more and more convinced that OAUTH is the right solution. What I need is a OAUTH server which fits into my GlassFish, WildFly JEE Server environments. – Ralph Sep 15, 2015 at 21:02

Great. I think you are making the right move, especially considering that your project is a FOSS. – Akira Sep 15, 2015 at 21:10

@Ralph, sorry for pointing that out, but I would not mind getting that green thing to show appreciation for my answer. :) – Akira Sep 15, 2015 at 21:25

3

Your question is about two independent issues.

Making your service accessible from another origin is easily solved by implementing CORS. For non-browser clients, cross-origin is not an issue at all.

The second problem about service authentication is typically solved using token based authentication.

Any caller of one of your microservices would get an access token from the authorization server or STS for

that specific service.

Your client authenticates with the authorization server or STS either through an established session (cookies) or by sending a valid token along with the request.

Share Improve this answer

Follow

answered Sep 15, 2015 at 5:01

MvdD

23.4k ● 9 ● 70 ● 97