Article
Heloisa Paiva · Jan 16, 2023 7m read

Token Authentication - Basics you need to start coding

Why I decided to write this

Recently I had the challenge to create a secure authentication method to authorize access to some data, but unfortunately I had zero experience with those security configurations and I felt that I was missing some basic concepts to have a better understanding of the official documentation.

After studying and managing to deliver the classes that I was asked to develop, I'd like to share a little bit of my new knowledge, which helped me follow the topics in the documentation.

Starting with the basics: the holy trinity of servers

First, it's important to understand what exactly we're dealing with. In general, we have some data that may be sensitive, or for any kinds of reason needs to be protected. There are some people (users) that will be able to see them, some will be able to change them, some won't have any kind of access. To take care of the users, access and data, we will have three servers: the client, the resources and the authorization.

So basically, here's the flow:

The client server sends a request to the authorization server, which will do the validation - checks if it can authorize it or not - which I will get into the details later. Once the server decides it is ok to allow access, it sends back to the client a token. The client now can send this token to the resources server, which will recognize it and return to the client the information it needs.

To be sure everything is clear before we move to next step, let's put ourselves in each server's shoes:

- **I'm the authorization server:** I want to recieve information from a client, and if I recognize this client, I'll provide him the key (the token) to access the data. I'm the concierge.
- **I'm the resources server:** I want to recieve a token. If this token is valid, whoever sent it will be able to access the data I guard. I'm the bank.
- **I'm the client server:** I want to access the data from the resources server, but I don't know the key. So I'll identify myself to the authorization so he'll give me the key. Then, I'll show my key to the resources server, who'll take me to the data I need. I'm the customer.

Going further: the token and the validation

But what is this validation and this token? Well, it is simple. The token is a string, which the resource server will interpret as the kind of access this client can have to the data. So if I have view access, my token will tell the resource server "She can perform a SELECT!" - and some other things like who am I, for how long can I use this token, etc - or if I have any kind of access, the token will gossip it all about me to the resource server and will tell it "She can perform anything she wants in this table!"

In practice, the authorization server will use some kind of encryption to transform "Hey, I'm Heloisa, my password for this server is PinkySwearImHonest.123" into "iuiDBIldb3287$@*++==" or something similar, which means "Resources server, give whoever sent you this, SELECT access" or any other kind of access that corresponds. The token could easily be a string saying literally "SELECT ACCESS", but that would have no security guaranteed.

But it's not all roses, so let's complicate it a bit. Of course the authorization server won't provide this token to anyone who asks, because you have to be very very special to access this data, right? To solve that, this server has a simple table with usernames, corresponding passwords and allowed access. The client has to indentify itself with a username and a password, so a validation can be performed. The validation is literally checking the table for usernames and passwords. The part we'll actually complicate a bit is how the client will send this information.

There are some methods:

- The client sends a request "Hey auth, I wan't to identify myself", and the authorization server prompts the user for the username and password. After performing the validation, it returns an authorization code, which the client will send again to the authorization server to get the token needed for the resources server.
- The same flow happens, but instead of getting an authorization code, the client will recieve the token directly;
- The client sends the username and password in a request to the authorization server, which will perform the validation and send back the token;
- There is no user, the client sends a Client ID and a Client Secret to obtain the token.

Each method has pros and cons, depending to the level of security the data needs. We'll discuss them later.

A little comparison to get things clearer

It kinda gave me the image of Hagrid and Harry Potter entering Gringotts Bank to get the Philosopher's Stone. Except that Dumbledore is the client server, who sends Harry, the user, and Hagrid, the client's request, saying "Hey, I need some data from the safe". The safe is the resources server, and the goblins are the authorization server, validating whether Hagrid can get the data for Dumbledore or not. And, of course, the key for the safe is the token. If you didn't watch Harry Potter you're probably way more confused than in the beggining, but now you can put the movie on thinking "this is for study, I'm not procrastinating". This is quite a good comparison actually, because Harry only knows what he's getting from the safe once he opens it, which is the whole point of the authentication. Also, Harry doesn't do anything except saying "Hi, I'm Harry Potter and I have no idea of what's going on, but somehow I got access to a very very rich safe that happens to be mine!". Notice how for the Dumbledore's lock he had SELECT access, because it was ok for him to see what was inside, but not to do anything with it. For his own lock, he had ALL access, because he could withdraw or deposit things, he could allow access for people he trusted, etc...

Maybe for the third method, he would've gone with Hagrid but would go shopping on Weasleys' Wizard Wheezes while Hagrid gets the package, and for the last one Dumbledore would've sent Hagrid alone to the bank.

Enough of Harry Potter, let's get back to business.

Getting hands dirty: the practice

Now that we have all the concepts clearer, we need to get the thing working. To start, you want to ask yourself "which server am I developing?".

If you're developing more than one, it might be helpful to do one at a time. Answering this question will lead you to focus on what exactly this class needs to do:

- **Resources server:** recieve a request with a token and check if this token is valid. If it is, perform the action the request sent (for example, SELECT * FROM db.table) and respond with the information obtained.

- **Authorization server:** depends on the type of validation chosen. For the first one, if it recieves an empty request, prompts the user for username and password. After that, performs the validation and if it's ok, responds with an authorization code. If it recieves a request with a valid authorization code, responds with the token. For the second one, recieves an empty request, prompts the user for username and password, performs the validation and if it turns out ok responds with the token. For the third, recieves a request with a username and password, performs validation and if it's ok responds with the token. For the last, recieves a request with a Client ID and Client Password, performs validation and if it's ok responds with the token.

- **Client server:** depends on the type of validation the authorization server uses. For the first one, sends a request to the authorization server and if the user has access, recieves a code in the response, sents this code in another request to the authorization server and recieves a token. Finally, sends this token to the resources server and recieves data. For the second one, sends a request to the authorization server if the user has access, recieves a token in the response, sends the token in a request to the resources server and recieves data. For the third, it recieves a username and password (it could be from another server, or prompting the user, or it could be a username and password fixed, etc), sends it in a request to the authorization server and if the user has access, recieves the token and blablabla to the resources and back with the data again. For the last one, it has a Client ID and Client Secret that will be sent in a request to the authorization server, it will recieve the token in the response and - blablabla again.

Discussing the validation methods

Most part of the security is due to the validation method. If you recieve specifics orders to develop in one or other kind of validation, you don't need to worry about this, but it is interesting to know why your boss made this choice. So if you're the one who needs to choose, or you're just interested, here are some good discussions about token authentication:

From StackOverflow: Proper way to send username and password from client to server

From StackExchange:Why shouldn't my client just send the user's username and password with every request?

PS.: I won't discuss them myself because it is out of the scope of this article, and also I think those links are worth reading in full.

The End!

Thank you very much for reading and I hope this article was helpful somehow.

Feel free to comment for doubts or contact me to help in specific cases or just to discuss. I'll be happy to be in touch!

#Access control #Authentication #Innovatium #InterSystems IRIS