OAuth
protocol

OAuth protocol

Prepared, edited, and published by *Jinan Kordab*, Programmer

[illegible]

My phone made two beeps, silent, with only vibrator mode on, and shut down. First beep was a long one, and the second shorter. It reminded me of a Terminator movie, where terminator, at the end of the movie, with one red eye, drowns slowly, into a pool of lava.

OAUTH protocol, when zoomed down with Shwarzenegger's eye, is very much flexible and versatile, which leaves room for design and creativity, while at the same time, keeping that strong idea of "WHO ARE YOU", and "WHAT ARE YOU ALLOWED TO DO HERE" intact.

So I have read and crunched each single page of OAUTH specification. About 77 pages in total, and the more I thought about it, the more it intrigued me. Many questions came up, some of which I will bring up here.

For example: "How does Authorization Server know Authentication Server or resource server and its refresh tokens ? Are they shared between them or stored somewhere ? Since with access token , you can perform some action and do some things after OAUTH completes."

Well, I have discovered that it is much more simpler than that. Generally speaking, you can design OAUTH, the way you want. But if you move away from the original specification, it will not be called OAUTH protocol at all. Cutting it short, it turns out that Authorization server does not need to know anything at all about access tokens, that a resource or Authentication server issues to me. Tokens are not shared between Authorization and Authentication or resource servers. Tokens are only issued by Authentication or resource server for me to be able to use its or some other resources.

Moreover, the Authorization Server, when processing first OAUTH request from me, acts as a function or method, with a set of parameters. Some are required, and others are optional, depending on one's OAUTH design. So in reality, my first call to Authorization server, is like a function call or a method call, where "response_type" and "client_id" parameters are required, while others are optional.

Keep in mind that at this stage, or at first request, no one knows of any token whatsoever !

Notice the "client_id" request parameter.

Here it is very important to note that one must first fill a registration form, manually or automatically, with Authorization Server, where it will store your registration details, and Client Id, which is given to you once you fill out a real world registration form.

I guess what makes OAUTH protocol strong, is its many forms of security joined together. Ex:

1. Manual real world form filling registration, where credentials and your issued ID are stored on Authorization Server.
2. Having another server called Authentication or Resource Server, that also asks you for certain credentials.
3. Choice of using all available cryptographic techniques to hash and secretize the response details that come from Authorization Server, and later used by Authentication or resource server.
4. Efficient use of computer clock, and time intervals to wait for certain parameters to expire, sent back from Authorization Server, and once expired, cannot be used to get a token from Authentication or resource server.
5. Optional parameters, if implemented in full, add additional layer of security.

So as you noticed, up till now, after sending a request to Authorization Server, with required parameters, "response_type" and "client_id", we should wait for a response. While waiting, and by good design, Authorization Server must return an error if those two required parameters are not found.

Et voila! We have a response from Authorization Server. Here, at this point, another question arises, and it is very important question: "Why does not Authorization Server automatically redirect our request to Authentication or resource server, and without sending a response to us? And in its turn an Authentication Server automatically issue access token?"

The answer is also very simple. OAUTH says that you can design and implement OAUTH this way also, and it is called "IMPLICIT" or "AUTOMATIC GRANT", and it is set in first request parameter "response_type", to Authorization Server, by setting its value to "token", thus a client directly receives an access token from Authorization Server. But if "response_type" value is set to "code", we receive the response from Authorization Server, but without access token.

So as you can see, access token is not something that is shared between the two servers, or stored on both servers. It can even be generated on the fly.

So finally we receive a response from first server (Authorization Server), with again, some required parameters. One of the required returned parameters is “code”, which we can and must use only once, and which expires shortly (this is the best option for the best results in OAUTH implementation). Another parameter is “state”, which is also required, and it holds original value sent from client to Authorization Server on first request. They should match !

Here are two things worth mentioning:

1. The “state” response parameter is very important, and is ignored in many OAUTH implementations.

Now, we are approaching the finish line. So far, we have a response from Authorization Server with following data:

- Code: (value)(expires shortly)(used only once)
- State(original value sent by us)(we can compare the values, and if different , throw an error)

We must ignore all false and unrecognized parameters, or after performing the comparison, exit with false, for “state”. “State”, at this stage is compared by us.

With “code” and “state”, we will get now an access token. So where do we send our request ? We send it to another, already mentioned before server called Authentication or Resource Server, with the following parameters:

- Grant_type (required)
- Code(required)
- Redirect_uri(optional)
- Client_id(optional – RED FLAG SECURITY IN SOME OAUTH IMIPLEMENTATIONS !!!!!!!!!!!!)

So we proceed and specify grant type, and say, for example, we want “implicit grant”, where we get access token, among other things. And with this token, go ahead and access, and start using resources, such as playing game A, adding a friend to a list of close friends, saving a document on a cloud, adding another email account to our email database, start backing up or replicating a database, fill some kind of unique form or a receipt, etc.., in short, all what we can do and use the resources.

The danger lies in “client_id” parameter mentioned before. We must send our “client_id” parameter, to Authentication Server or resource server, to prevent mixing it with “code” generated for another client !!! I saw a very famous website has this loophole! It is a very famous website. They have OAUTH implemented. You need to create an app on their site, register it with them officially, log in to your app with your credentials, using their API, which in turn uses OAUTH, and what happens is that in response, you get logged in as a different user, under someone else’s name ! This is live example of false OAUTH implementation, and NOT SENDING THE CLIENT ID TO AUTHENTICATION SERVER !

So, the answer to our first initial question would be that “client_id” should be shared or somehow exchanged between the two servers, Authentication Server and Authorization Server, for correct and strong OAUTH implementation.

The response we get after sending “grant_type”, “code”, “redirect_uri” and “client_id” to Authentication Server is this:

- Access_token (required)(throw an error if not present)
- Token_type(required)(case_sensitive)(throw an error if not present)
- Expires_in(recommended)
- State(should be the same one that we sent in the beginning, in order to compare both)(If not the same, throw an error)

A side note: When refreshing a token, having both Required and Optional parameters, that we are going to send to Authentication Server , the refresh token “scope” parameter value must be identical to that of refresh token included by the client on first request for refresh token.

Also, when using implicit grant type flow, a refresh token is not returned, which requires repeating the authorization process and speaking again to first server (Authorization Server). Implicit grant type means you get redirected from Authorization server to Authentication Server without getting first response from Authorization Server.

Thank you,

Jinan Kordab

Programmer