Oct 31st, 9:00 AM - 10:30 AM

OAuth2.0 in Securing APIs

Olimpion Shurdi
University of Tirana

Aleksander Biberaj
University of Tirana

Igli Tafa
University of Tirana

Genci Mesi
University of Tirana

OAuth2.0 in Securing APIs

Olimpion Shurdi [1], Aleksander Biberaj[2] , Igli Tafa[3] , Genci Mesi [4]

[1234]Faculty of Information Technology, Polytechnic University of Tirana,

Tirana, 1001, Albania

Abstract: Today's modern applications are mostly designed around API's. API's are used for a variety of things such as passing data to another webservice reading data from a database etc. The problem with this is that not all the API's are secure. Most of the today's API's are old and rely only on an authentication token where the user data often had to share their credentials with the application to enable such an API call on their behalf or string them, which is often hardcoded. We will focus on OAUTH 2.0 as new protocol in securing our API's. This is a new protocol based on delegation of authorization, dynamically changing authentication string based on user session or application session. We will go on this different mode of authentication and show you how to use them properly. We will set up this with a Web API integrated with OAUTH and a client application that will stimulate the requests to our API's.

Keywords: API, REST, authorization, OAUTH, security, open platform.

1. Introduction

Many of the modern web application developed by companies like HR, accounting, billing, invoices, or any other system today. This developed software's or microservices needs to communicate with each other for various reasons like being built on different platforms, different programming languages etc. That is why we need a universal way of communicating with this different software's as the software scales.That is where the API were born to fill this gap in the software industry. This technology let us the opportunity to execute tasks or programs inside another program without the need of the human interaction.The oldest ways of the implementation of an API was with WSDL or SOAP or REST which are basically based on layer 7 to execute their data transfers.API adoptions today is everywhere today ranging from usage on personal projects to consumers and enterprises with many of the big name's companies being involved and offering services to facilitate building of an API.This big names are doing their development by alternating the kernel of the API to be extended by the same API using models and component based GUI Web applications.This is very useful in today's large scale application targeting the end user platforms like mobile,Windows, Mac or any Linux distribution[1].

Basic logic of the API functionality is shown in Figure l.

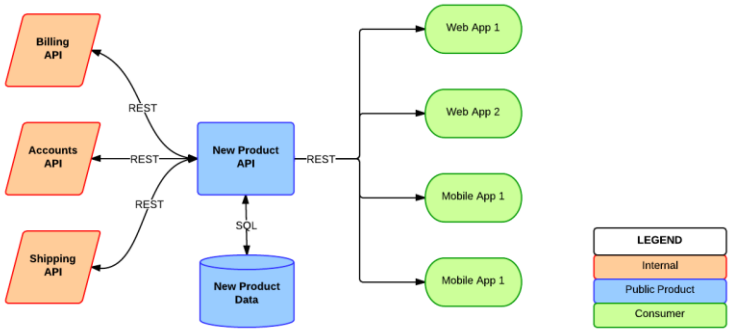

Figure 1. API logic example

1.1 How can we secure an API

The API changed the way we think about data and its hierarchy. An API can basically serve as a client to another web application or as a server to another web application or serve as third-party for two or more applications. This brings to life old challenges like complexity, speed, authentication, and authorization. In the old way of implementation the user which serves as a client would use its credentials for the authentication process, so he had to share this credentials with a third party application and this posed some limitations and problems like the ones listed below:

- Third-party applications store these credentials in the browser for future use often in clear text.
- Servers in their logic. required a password for the API authentication and this created a new problem for other technologies like Captchas, multifactor authentication to further secure user's account.
- Client applications gained unrestricted access to all the user's data and subsets without giving him the opportunity to limit it in any way.
- Application linked with each other did not give the opportunity to the user to limit a certain application access because by doing so he will limit all the applications.
- Corrupted applications will result in corrupting all the used data, password, and subsets.

That's why the big names in the industry today like Google,Amazon,Yahoo, Aliexpress etc. abandoned this simple implementation and worked on providing their own implementation, switching from the old implementation to a delegated authorization inside the specified service granted by user to the client application [2].

This implementation of Google is shown in Figure 2.

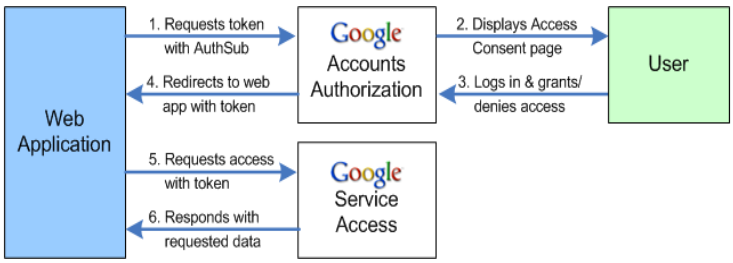

Figure 2. Google's implementation of AuthSub flow

The implementation of Yahoo is shown in Figure 3

Figure 3. Yahoo implementation of BBAuth flow

BBAuth of Yahoo was the first implementations that is near OAuth today while AuthSub provided by Google was considered deprecated and went out of service in August 2012 officially [3]. Differencies between BBAuth and OAuth is that OAuth supports all the software applications ranging from web, desktop and mobile while BBAuth supported only web applications.This new protocols in the backend, what they basically did is they redirected Api requests to the authenticatior page to verify the requests parameters. After it was verified the application itself would release a token compatible to itself and the authenticator provider so that the user could access its data.This way of' implementation lead all the software industy from startups to the big companies to use this Api providers without going themselves to build the Api fearing it would expose any vulnerability in their data transfers.

1.2 OAuth 1.0

OAuth is a widely used protocol or framework for authorization. It provides a secure way for clients to access private service resources that need authorization without sharing the client's real data/credentials. The way it works is that it does not send back the client's real data (like password). Instead, it sends send back a token as a response. The token is a string which contains data, like the identity of the consumer. The client then uses this token as a payload in every request it sends to the service. The services validate and verifies the identity of the request. This way, no private data travels throughout the internet making OAuth a secure way of authorization.

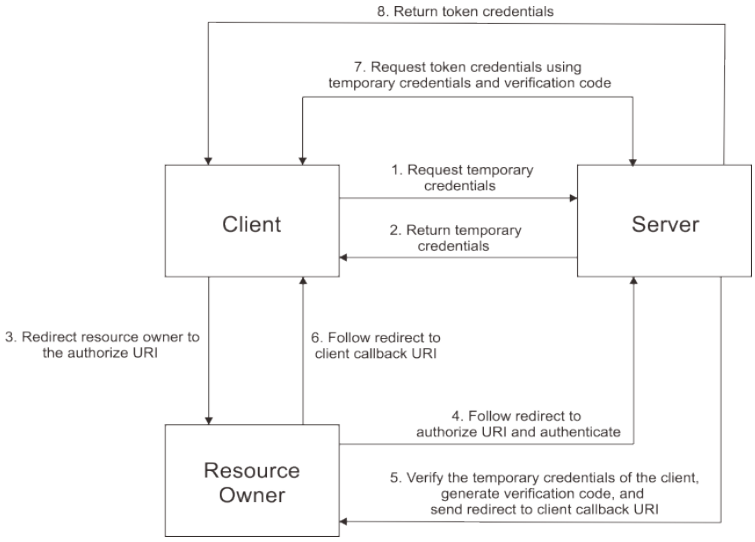

Figure 4. OAuth version 1 .0 flow

One thing to note is that OAuth is built on previously existing protocols created independently by various companies. It is built with the purpose to unify all the best practices of these previous solutions into a single community-driven implementation design. By being community-driven, it promotes standard and consistent development for both services and client applications.

OAuth 1.0 required knowledge on cryptography. Every request must have a cryptographic. signature as payload in order to verify and authorize the request, but cryptography has always been a very challenging topic to master, even for the most skilled software engineers. That is why, its implementation/adoption encountered in difficulties since authorization back then could be achieved with other easier protocols.

2. OAuth version 2.0

OAuth is the next implementation of the evolution of OAuth protocol .This will lead to a new change of Api need to be done by every company that used OAuth version 1.0 in order to upgrade to the new version offering new advanced security features but not being backwards compatible with the old protocol[4].

The new version focused on simplicity on client's development while still providing specific flows in terms of security, authorization and authentication in all devices connected to the internet. New features include this flow on IOT devices.

All this changes are done and documented by fETF Auth group and you can trace to the original documentation if you want dating back to 2012.

OAuth allows to share credentials with other sites you frequently use to save username and password.

This is a out of box implementation in all of our gmail accounts or you can use any of the chrome .Mozilla plugins like Last Pass etc.

A simplified implementation of this protocol is shown in Figure 5.

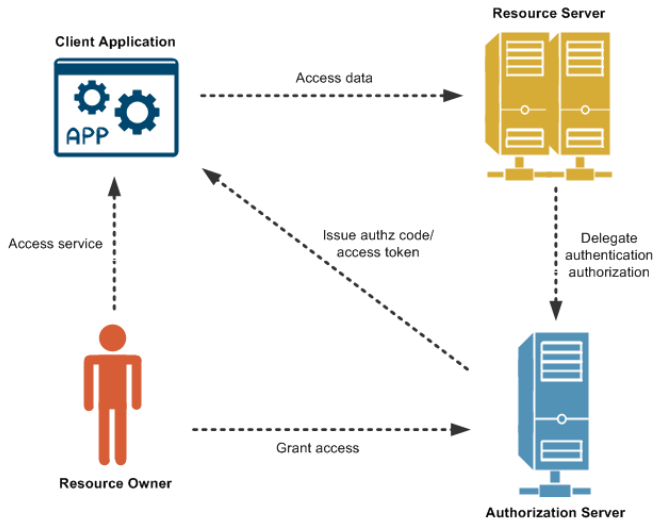

Figure 5. OAuth 2.0 simplified scheme

The new feature of this protocol is that you do not need to worry anymore for which one the cryptographic algorithms to use in your application. You just need to add TLS to your application, and this will handle all the encryption for you.

2.1 Advancements from OAuth version 1.0

OAuth version 1.0 was widely use and based upon the implementations we showed in the first section of our paper like BBAuth and Auth Sub .This protocols lead the foundation and trained the community and triggered it to rethink on ways to improve it on the three areas we will list below:

- Authentication and Signatures

The biggest annoyance for everyone in the software industy using this protocol was that the cryptographic algorithms required as input the client identifier and secret. This is no more an issue with version 2.0 which needs to use Https for every operation with the Api otherwise it wi II give you a 400 bad request code.

- UX and alternative authentication application nows

OAuth in its basic principle require one input which is the access token to continue to send the requests.As the variety of the devices we use every day with access to the internet grew exponentially having a large number of users today using smart tv, consoles .Iot smart devices etc OAuth version 1.0 failed to deliver the best user experience to these end user devices entering the market.OAuth version 2.0 extends the base protocol usability and user experience by providing long term compatibility with future inventions and devices.

- Performance at Scale

As all the industy embraced the version 1.0 of OAuth it was noted that this protocol in large application did not scale well expecially on data centers which needed to make workaround like providing temporary states and credentials. It also required the applications id and password but having a complex infrastructure would break it making them to use different separated Api servers. This implementation is fixed in version 2.0 which now supports hierarchy of roles in obtaining authorization and continuing in handling Api calls. This gives the chance to use the same server for many roles[6],[7].

2.2 User roles and application flows.

OAuth as a security protocol defines 4 major roles which we are going to talk in this section in details.

- Authorization Server

The server's handles delivering the access tokens and authorization codes to the client applications in order for them to user's data in the resource's server.

- Resources Server

The resources server is represented as a server where we host all of our user's protected and encrypted data for all our applications. A typical example is an Api provider that keeps and protects data.

- Client Application

The client applications is the application we are going to server the user's data as a response to their Api calls. The client will also need the user's permission to continue with this operation.

- User as owner of the resources

This is represented by the end user itself. It is the person who will provide the access to all the data or a portion of its data to the application.
Below in Figure. 6 you'll find an implementation of the OAuth version 2.0 logic widely used today by the modern web applications [8],[9].

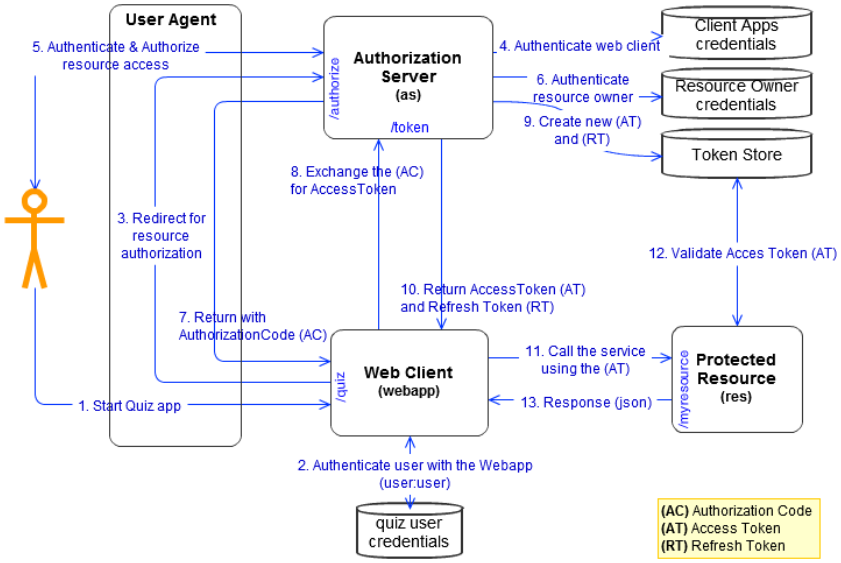

Figure .6 Authorization flow for OAuth version 2.0

2.3 Types of grants

OAuth 2 .0 is a very flexible framework since it provides a number of ways for thtFJent to obtain an access token.Each provided type can be used for different cases. The grant types for providing an access token are:

- Authorization Code

This authorization type is most used for web applications. Web applications are applications written in a server-side language and hosted on a web server. Their code is hidden from the public.

- Implicit

This authorization type is most tit for browser-based or mobile applications. A browser-based application is rm application in which the whole source code is loaded from a web pag.d run on the browser entirely. A typical browser-application is built on Javascript, React JS etc. This method, instead ofreturning an authorization code, returns an access token which will be used as a payload on every request. Since it resides on the browser it is not entirely secret.

- Password

This method requires entering "username" and "password" directly to get a token back. Since it requires collecting the password, it should only be used by client applications which are owned by the service owner itself', for example: the same person or company ows both the service and the client application. So, this is a good method for trusted clients on the web or native device applications (desktop or mobile).

- Client credentials

This method is suitable for authentication where a certain user's permission is not required. It is used when an application requests access to its own resources on behalf of itself, not on any user. To be more precise, the client requests for a token using only the client credentials. No additional request parameter is needed.

3. Code Implementation

To stimulate the workflow of the OAuth version 2.0, we used PHP as the programming language to build our Api server with build-in features of OAuth version 2.0. To test the client side, we also created an Api client application to use the Authorization Code as grant type.
The workflow is explained in the following steps:

1. The client application will redirect the user to our newly created Api server. It will serve as the as the Authorization server for our client application to access data inside.
2. After the user approve access (detailed in Figure .7) the client application will receive a callback function containing an authorization code.
3. After the authorization code is. obtained the cl rents application will use this as the input to generate the access token.
4. After the authorization code is validate, our Api server will pass the access token as a response to the client application.
5. Now that we have the access token, we can move safely to request the protected data. The client calls for this data to the server.
6. After the access token is validated it will send back a full request with the containing protected data in it to our client application.

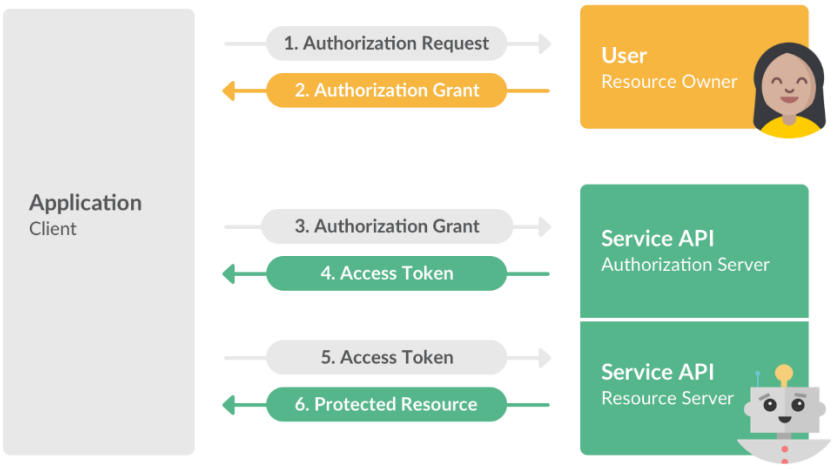

Figure 7. Client form to confirm Authorization approval

4. Conclusion

A lot of people in the software industry found that version 2 .0 of this protocol was not that safe by default. That's why it was considered more of ‘a framework than a protocol by itself. That's why most of this protocol implementations me customized specifically for the application needs.
The main problems pointed out by the experts of the field related to the OAuth v2.0 is that there is a delicate line between security and usability,
Despite these drawbacks, OAuth version 2.0 is one the most used implementation today by the application developers.
Because of its design and ease-of-implementation it has become a standard for building other protocols.

References

1. "OAuth core group" 'Uffi.cia] OAuth vl.O" https://oauth.netloot·e/1.0/
2. E. Hamer-Lahav The OAuth].0 Protocol' "https://tools.ietf.org/html/rfc5849'
3. A. Parecki, "An Introduction to OAuth 2," in *Esri International User Conference*, California, 2013
4. Synopsys Editorial Team" What's the difference? OAuth LO vs OAuth 2.0" https://www. synopsys.corn!blogs/software-secur]ty/oauth-2-0- vs-oauth-Lfl/
5. OAuthl Overview' 'Soup Ul' 'https://www. saapui.org/docs/oauthl/oauthl - overview.html'
6. D. Hardt" The OAuth 2.0 Authorization Framework" 'https.z/tools .ietf.org/htrn.l/rfc6749"
7. OAuth Docs" iAuth 2.0 Authorization" https://auth0.com/docs/protocols/oauth2
8. "Matt Raible" "What the Heck is OAuth?" https://deve]oper.ok:ta.com/blog/20l7/06/21/what-the-heck-is-oauth"
9. Aaron Parecki 'OAuth 2 Simplified" 'https:l/aaronparecki.com/oauth-2-simplifie