(/)

HOME (/)

COURSES (/COURSES)

The Ins and Outs of Token Based Authentication

Chris Sevilleja (/@chris) (@chrisoncode) (https://twitter.com/chrisoncode)

January 21, 2015

#authentication (/tutorials?hFR%5Bcategory%5D%5B0%5D=Tutorials&dFR%5B_tags%5D%5B0%5D=authentication)

#tokens (/tutorials?hFR%5Bcategory%5D%5B0%5D=Tutorials&dFR%5B_tags%5D%5B0%5D=tokens)

391,660 views

88 comments

LOGIN (/LOGIN) SIGN UP (/REGISTERING)

(/@chris)
Chris Sevilleja
(/@chris)
Follow @chrisoncode 8,326 followers

January 21, 2015

#authentication (/tutorials?hFR%5Bcategory%5D%5B0%5D
#tokens (/tutorials?hFR%5Bcategory%5D%5B0%5D

88 COMMENTS

391,660 VIEWS

#Introduction

Token based authentication is prominent everywhere on the web nowadays. With most every web company using an API, tokens are the best way to handle authentication for multiple users.

There are some very important factors when choosing token based authentication for your application. The main reasons for tokens are:

Stateless and scalable servers
Mobile application ready

Pass authentication to other applications
Extra security

Table of Contents

#Who Uses Token Based Authentication?

Any major API or web application that you've come across has most likely used tokens. Applications like Facebook, Twitter, Google+, GitHub, and so many more use tokens.

Let's take a look at exactly how it works.

#Why Tokens Came Around

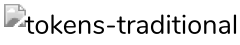
Before we can see how token based authentication works and its benefits, we have to look at the way authentication has been done in the past.

Server Based Authentication (The Traditional Method)

> Since the HTTP protocol is *stateless*, this means that if we authenticate a user with a username and password, then on the next request, our application won't know who we are. We would have to authenticate again.

The traditional way of having our applications remember who we are is to **store the user logged in information on the server**. This can be done in a few different ways on the session, usually in memory or stored on the disk.

Here is a graph of how a server based authentication workflow would look:



(https://cask.scotch.io/2014/11/tokens-traditional.png)
As the web, applications, and the rise of the mobile application have come about, this method of authentication has shown problems, especially in scalability.

#The Problems with Server Based Authentication

A few major problems arose with this method of authentication.

Sessions: Every time a user is authenticated, the server will need to create a record somewhere on our server. This is usually done in memory and when there are many users authenticating, the overhead on your server increases.

Scalability: Since sessions are stored in memory, this provides problems with scalability. As our cloud providers start replicating servers to handle application load, having vital information in session memory will limit our ability to scale.

CORS: As we want to expand our application to let our data be used across multiple mobile devices, we have to worry about cross-origin resource sharing (CORS). When using AJAX calls to grab resources from another domain (mobile to our API server), we could run into problems with forbidden requests.

CSRF: We will also have protection against cross-site request forgery (https://en.wikipedia.org/wiki/Cross-site_request_forgery) (CSRF). Users are susceptible to CSRF attacks since they can already be authenticated with say a banking site and this could be taken advantage of when visiting other sites.

With these problems, scalability being the main one, it made sense to try a different approach.

#How Token Based Works

Token based authentication is **stateless**. We are not storing any information about our user on the server or in a session.

This concept alone takes care of many of the problems with having to store information on the server.

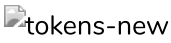
> No session information means your application can scale and add more machines as necessary without worrying about where a user is logged in.

Although this implementation can vary, the gist of it is as follows:

1. User Requests Access with Username / Password
2. Application validates credentials
3. Application provides a signed token to the client
4. Client stores that token and sends it along with every request
5. Server verifies token and responds with data

Every single request will require the token. This token should be sent in the HTTP header so that we keep with the idea of stateless HTTP requests. We will also need to set our server to accept requests from all domains using `Access-Control-Allow-Origin: *`. What's interesting about designating `*` in the ACAO header is that it does not allow requests to supply credentials like HTTP authentication, client-side SSL certificates, or cookies.

Here's an infographic to explain the process:



(https://cask.scotch.io/2014/11/tokens-new.png)
Once we have authenticated with our information and we have our token, we are able to do many things with this token.

We could even create a permission based token and pass this along to a third-party application (say a new mobile app we want to use), and they will be able to have access to our data -- **but only the information that we allowed with that specific token**.

#The Benefits of Tokens

Stateless and Scalable

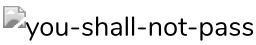
(https://cask.scotch.io/2014/11/infinity.jpg)
Tokens stored on client side. Completely stateless, and ready to be scaled. Our load balancers are able to pass a user along to any of our servers since there is no state or session information anywhere.

If we were to keep session information on a user that was logged in, this would require us to keep sending that user to the *same server that they logged in at* (called session affinity).

This brings problems since, some users would be forced to the same server and this could bring about a spot of heavy traffic.

Not to worry though! Those problems are gone with tokens since the token itself holds the data for that user.

Security



(https://cask.scotch.io/2014/11/you-shall-not-pass.jpg)
The token, not a cookie, is sent on every request and since there is no cookie being sent, this helps to prevent CSRF attacks. Even if your specific implementation stores the token within a cookie on the client side, the cookie is merely a storage mechanism instead of an authentication one. There is no session based information to manipulate since we don't have a session!

The token also expires after a set amount of time, so a user will be required to login once again. This helps us stay secure. There is also the concept of token revocation (https://tools.ietf.org/html/rfc7009) that allows us to invalidate a specific token and even a group of tokens based on the same authorization grant.

Extensibility (Friend of A Friend and Permissions)

(https://cask.scotch.io/2014/11/share-candy.jpg)

Tokens will allow us to build applications that share permissions with another. For example, we have linked random social accounts to our major ones like Facebook or Twitter.

When we login to Twitter through a service (let's say Buffer), we are allowing Buffer to post to our Twitter stream.

By using tokens, this is how we **provide selective permissions to third-party applications**. We could even build our own API and hand out special permission tokens if our users wanted to give access to their data to another application.

Multiple Platforms and Domains

We talked a bit about CORS earlier. When our application and service expands, we will need to be providing access to all sorts of devices and applications (since our app will most definitely become popular!).

Having our API just serve data, we can also make the design choice to serve assets from a CDN. This eliminates the issues that CORS brings up after we set a quick header configuration for our application.

JAVASCRIPT

```
Access-Control-Allow-Origin: *
```

Our data and resources are available to requests from any domain now **as long as a user has a valid token**.

Standards Based

When creating the token, you have a few options. We'll be diving more into this topic when we secure an API in a follow-up article, but the standard to use would be JSON Web Tokens (https://scotch.io/tutorials/the-anatomy-of-a-json-web-token).

This handy debugger and library chart shows the support for JSON Web Tokens. You can see that it has a great amount of support across a variety of languages. This means you could actually switch out your authentication mechanism if you choose to do so in the future!

#Conclusion

This was just a look at the how and why of token based authentication. As is always the case in the world of security, there is much, much, much, much (too many?) more to each topic and it varies per use case. We even dove into some topics on scalability which deserves its own conversation as well.

This was a high level quick overview, so please feel free to point out anything that was missed or any questions you have on the matter.

In our next article, we'll be looking at the anatomy of JSON Web Tokens (https://scotch.io/tutorials/the-anatomy-of-a-json-web-token). For full code examples on how to authenticate a Node API using JSON Web Tokens, check out our book MEAN Machine (https://leanpub.com/mean-machine).

Edit #1 Adding ACAO header info and CSRF clarifications (thanks to Emily Stark (https://twitter.com/estark37) for the article info)

Build Your First Angular Website (/courses/build-your-first-angular-website)

Getting Started with Scotch Box: Vagrant for the Absolute Beginner (/courses/getting-started-with-scotch-box-vagrant-for-the-absolute-beginner)

More Courses(/courses)

Latest Posts

Getting Started with JavaScript Object (/tutorials/getting-started-with-javascript-object)

Build a web traffic monitor with Python (/tutorials/build-a-web-traffic-monitor-with-python)

8 Effective Design Tips for Offline Applications (/tutorials/8-effective-design-tips-for-offline-applications)

How to use Error Boundaries in React 16 (/tutorials/how-to-use-error-boundaries-in-react-16)

5 Most Common Dropdown Use Cases Solved with React Downshift (/tutorials/5-most-common-dropdown-use-cases-solved-with-react-downshift)

(/tutorials?hFR%5Bcategory%5D%5B0%5D=Tutorials)

(/@chris)

CHRIS SEVILLEJA

(/@CHRIS)

Founder of Scotch.io. Google Developer Expert in Web Technologies. Slapping the keyboard until something good happens.

Follow @chrisoncode 8,326 followers

VIEW MY 176 POSTS (/@CHRIS)

1 88

Related Course
Getting Started with JavaScript for Web Development (https://bit.ly/2rVqDcs)

(/@chris)

Chris Sevilleja (/@chris)

176 posts (/@chris)
Founder of Scotch.io. Google Developer Expert in Web Technologies. Slapping the keyboard until something good happens.

oncode) (https://facebook.com/sevilayha) (https://github.com/sevilayha) (https://instagram.com/chriscode.chrislift) (https:

Popular In the Community

JAVASCRIPT FUNCTIONAL PROGRAMMING...	DEBUGGING NODE CODE IN VS CODE	UPGRADE ANGULARJS SORTING FILTERS TO...	8 EFFECTIVE DESIGN TIPS FOR OFFLINE...	BUILDING A FANCY COUNTDOWN TIMER...	BUILD NATIVE MODALS USING THE DIALOG...	PASSWORD VERIFICATION
CyanPaw 24 Jun	OrangeDucky 26 Jun	Daljeet Arora 21h	GreenDog 28 Jun	kjtxht 25 Jun	Craig Geil 25 May	OliveB 1 Jun
I would lose my shit if someone would...	nnjn	Great Source of information....	great article!	Amazing effect. And looks kinda simple o...	Will you be able to drag the dialog...	Nice!

(https://dynamic-cdn.spot.im/yad/optout.html)

A side project brought to you from Las Vegas and DC by... (/about)

(https://scotch.io/@chris)

Chris Sevilleja (https://scotch.io/@chris)

Follow @chrisoncode 8,326 followers

(https://scotch.io/@nick)

Nick Cerminara (https://scotch.io/@nick)

Follow @whatnicktweets 2,437 followers

(https://bit.ly/2tDcLEK)

Easiest Local Dev Environment

(https://scotch.io/courses/getting-started-with-vue)

Get Started with Vue.js

scotch

Top shelf learning. Informative tutorials explaining the code **and the choices behind it all.**



(https://github.com/scotch-scotchdevelopment)

(https://twitter.com/scotchdevelopment) (https://facebook.com/scotchdevelopment)