Cookies vs Tokens. Getting auth right with Angular.JS (/blog/2014/01/07/angularjs-authentication-with-cookies-vs-token/)

Using a token-based authentication design over cookie-based authentication.

 Alberto Pose (http://twitter.com/thepose)

January 07, 2014

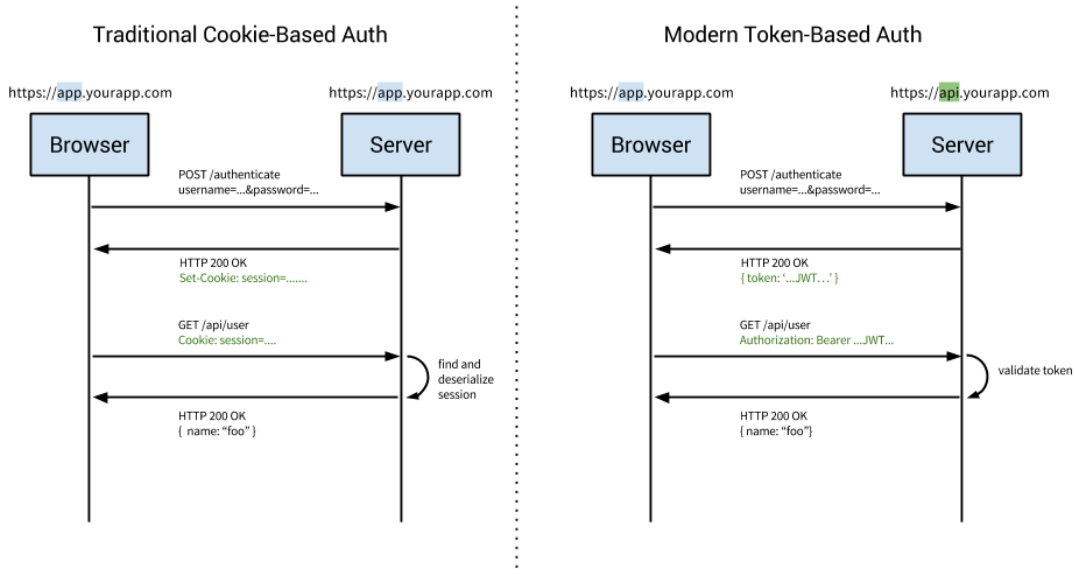
Introduction

There are basically two different ways of implementing server side authentication for apps with a frontend and an API:

> The most adopted one, is **Cookie-Based Authentication** (you can find an example here (http://frederiknakstad.com/authentication-in-single-page-applications-with-angular-js/)) that uses server side cookies to authenticate the user on every request.
>
> A newer approach, **Token-Based Authentication**, relies on a signed token that is sent to the server on each request.

Token based vs. Cookie based

The following diagram explains how both of these methods work.



(https://docs.google.com/drawings/d/1wtiF_UK2e4sZVorvfBUZh2UCaZq9sTCGoaDojSdwp
usp=sharing)

What are the benefits of using a token-based approach?

Cross-domain / CORS: cookies + CORS don't play well across different domains. A token-based approach allows you to make AJAX calls to any server, on any domain because you use an HTTP header to transmit the user information.

Stateless (a.k.a. Server side scalability): there is no need to keep a session store, the token is a self-contanined entity that conveys all the user information. The rest of the state lives in cookies or local storage on the client side.

CDN: you can serve all the assets of your app from a CDN (e.g. javascript, HTML, images, etc.), and your server side is just the API.

Decoupling: you are not tied to a particular authentication scheme. The token might be generated anywhere, hence your API can be called from anywhere with a single way of authenticating those calls.

Mobile ready: when you start working on a native platform (iOS, Android, Windows 8, etc.) cookies are not ideal when consuming a secure API (you have to deal with cookie containers). Adopting a token-based approach simplifies this a lot.

CSRF: since you are not relying on cookies, you don't need to protect against cross site requests (e.g. it would not be possible to `<iframe>`

your site, generate a POST request and re-use the existing authentication cookie because there will be none).

Performance: we are not presenting any hard perf benchmarks here, but a network roundtrip (e.g. finding a session on database) is likely to take more time than calculating an *HMACSHA256* (http://en.wikipedia.org/wiki/Hash-based_message_authentication_code) to validate a token and parsing its contents.

Login page is not an special case: If you are using Protractor (https://github.com/angular/protractor) to write your functional tests, you don't need to handle any special case for login (https://github.com/angular/protractor/issues/51).

Standard-based: your API could accepts a standard JSON Web Token (http://tools.ietf.org/html/draft-ietf-oauth-json-web-token) (JWT). This is a standard and there are multiple backend libraries (.NET (https://www.nuget.org/packages?q=JWT), Ruby (http://rubygems.org/search?utf8=%E2%9C%93&query=jwt), Java (https://code.google.com/p/jsontoken/), Python (https://github.com/davedoesdev/python-jwt), PHP (https://github.com/firebase/php-jwt)) and companies backing their infrastructure (e.g. Firebase (https://www.firebase.com/docs/security/custom-login.html), Google (https://developers.google.com/accounts/docs/OAuth2ServiceAccount#overview), Microsoft (https://github.com/MSOpenTech/AzureAD-Node-Sample/wiki/Windows-Azure-Active-Directory-Graph-API-Access-Using-OAuth-2.0)). As an example, Firebase (https://www.firebase.com/docs/security/custom-login.html) allows their customers to use any authentication mechanism, as long as you generate a JWT with certain pre-defined properties, and signed with the shared secret to call their API.

> *What's JSON Web Token? **JSON Web Token (JWT**, pronounced jot) is a relatively new token format used in space-constrained environments such as HTTP Authorization headers. JWT is architected as a method for transferring security claims based (http://en.wikipedia.org/wiki/Claims-based_identity) between parties.*

Implementation

Asuming you have a node.js app, below you can find the components of this architecture.

Server Side

Let's start by installing `express-jwt` and `jsonwebtoken`:

```
$ npm install express-jwt jsonwebtoken
```

Configure the express middleware to protect every call to `/api`.

```
var expressJwt = require('express-jwt');
var jwt = require('jsonwebtoken');

// We are going to protect /api routes with JWT
app.use('/api', expressJwt({secret: secret}));

app.use(express.json());
app.use(express.urlencoded());
```

The angular app will perform a POST through AJAX with the user's credentials:

```
app.post('/authenticate', function (req, res) {
  //TODO validate req.body.username and req.body.password
  //if is invalid, return 401
  if (!(req.body.username === 'john.doe' && req.body.password === 'foobar')) {
    res.send(401, 'Wrong user or password');
    return;
  }

  var profile = {
    first_name: 'John',
    last_name: 'Doe',
    email: 'john@doe.com',
    id: 123
  };

  // We are sending the profile inside the token
  var token = jwt.sign(profile, secret, { expiresInMinutes: 60*5 });

  res.json({ token: token });
});
```

GET'ing a resource named `/api/restricted` is straight forward. Notice that the credentials check is performed by the `expressJwt` middleware.

```
app.get('/api/restricted', function (req, res) {
  console.log('user ' + req.user.email + ' is calling /api/restricted');
  res.json({
    name: 'foo'
  });
});
```

Angular Side

The first step on the client side using AngularJS is to retrieve the JWT Token. In order to do that we will need user credentials. We will start by creating a view with a form where the user can input its username and password.

```
<div ng-controller="UserCtrl">
  <span></span>
  <form ng-submit="submit()">
    <input ng-model="user.username" type="text" name="user" placeholder="Username
    <input ng-model="user.password" type="password" name="pass" placeholder="Pass
    <input type="submit" value="Login" />
  </form>
</div>
```

And a controller where to handle the submit action:

```
myApp.controller('UserCtrl', function ($scope, $http, $window) {
  $scope.user = {username: 'john.doe', password: 'foobar'};
  $scope.message = '';
  $scope.submit = function () {
    $http
      .post('/authenticate', $scope.user)
      .success(function (data, status, headers, config) {
        $window.sessionStorage.token = data.token;
        $scope.message = 'Welcome';
      })
      .error(function (data, status, headers, config) {
        // Erase the token if the user fails to log in
        delete $window.sessionStorage.token;

        // Handle login errors here
        $scope.message = 'Error: Invalid user or password';
      });
  };
});
```

Now we have the JWT saved on `sessionStorage`. If the token is set, we are going to set the `Authorization` header for every outgoing request done using `$http`. As value part of that header we are going to use `Bearer <token>`.

> `sessionStorage`: *Although is not supported in all browsers (you can use a polyfill (https://github.com/inexorabletash/polyfill/blob/master/storage.js)) is a good idea to use it instead of cookies (`$cookies`, `$cookieStore`) and `localStorage`: The data persisted there lives until the browser tab is closed.*

```
myApp.factory('authInterceptor', function ($rootScope, $q, $window) {
  return {
    request: function (config) {
      config.headers = config.headers || {};
      if ($window.sessionStorage.token) {
        config.headers.Authorization = 'Bearer ' + $window.sessionStorage.token;
      }
      return config;
    },
    response: function (response) {
      if (response.status === 401) {
        // handle the case where the user is not authenticated
      }
      return response || $q.when(response);
    }
  };
});

myApp.config(function ($httpProvider) {
  $httpProvider.interceptors.push('authInterceptor');
});
```

After that, we can send a request to a restricted resource:

```
$http({url: '/api/restricted', method: 'GET'})
.success(function (data, status, headers, config) {
  console.log(data.name); // Should log 'foo'
});
```

The server logged to the console:

```
user foo@bar.com is calling /api/restricted
```

The source code is here (https://github.com/auth0/angular-token-auth) together with an AngularJS seed app.

What's next?

In upcoming posts we will revisit:

- How to handle social authentication?
- How to handle session expiration?

UPDATE: we published two new blog posts

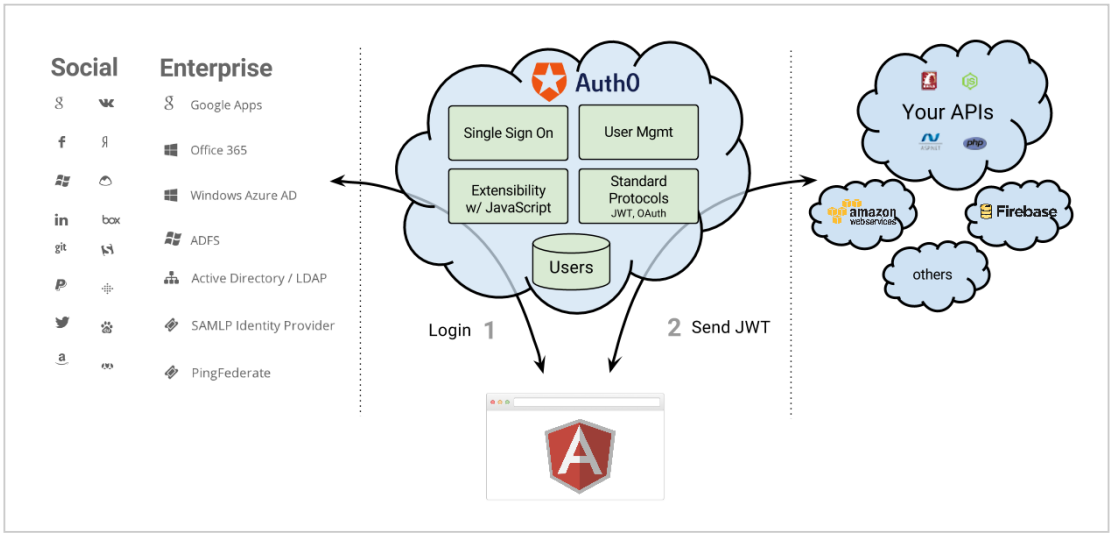
- Token based authentication in realtime frameworks like Socket.io (/blog/2014/01/15/auth-with-socket-io/)
- 10 Things you should know about Tokens (/blog/2014/01/27/ten-things-you-should-know-about-tokens-and-cookies/)

Bottom Line

When building Single Page Applications, consider using a token-based authentication design over cookie-based authentication. Leave a comment or discuss on HN (https://news.ycombinator.com/item?id=7018529).

Aside: how it works with Auth0?

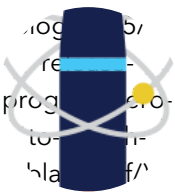
Auth0 issue **JSON Web Tokens** on every login. That means that you can have a solid identity infrastructure, including **Single Sign On**, **User Management**, support for **Social**, **Enterprise** and **your own database** of users with just a few lines of code. We implemented a tight integration with Angular: https://github.com/auth0/auth0-angular (https://github.com/auth0/auth0-angular)



(https://docs.google.com/a/auth10.com/drawings/d/1ErB68gFj55Yg-ck1_CZByEwN5qlOPj2Mzd-6S5umv20/edit)

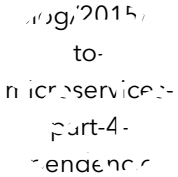
More about Auth0 and Angular: https://github.com/auth0/auth0-angular (https://github.com/auth0/auth0-angular)

RECENT POSTS



"Pre-release Program, Zero To Launch, Blasts Off!" (/blog/2015/11/10/pre-release-program-zero-to-launch-blasts-off/)

Introducing angular2-jwt: A Library for Angular 2 Authentication (/blog/2015/11/10/introducing-angular2-jwt-a-library-for-angular2-authentication/)



Intro to Microservices, Part 4: Dependencies and Data Sharing (/blog/2015/11/07/introduction-to-microservices-part-4-dependencies/)

Featured Comment

José Romaniello Mod • a year ago

Hi everyone subscripted to this comment thread. We appreciate very much your interest for this technology, the questions and answers.

We will be closing the discussion here because it is very hard to follow and it is not easy to search.

Feel free to open specific topics regarding JWT in our discourse here:

https://ask.auth0.com/category...

Thanks again!

3 ^ | v • Share ›

Comments for this thread are now closed. ×

110 Comments Auth0 Blog 1 Login

Recommend 21 Share Sort by Best

Darqul • a year ago

Let me ask you something.
How is this different from a security perspective than what cookies are doing?
On every request you send an "Authorization" header, with a cookie on every request you're sending a "Cookie" header.
sessionStorage or localStorage in both a document with an iframe can read the values stored in them. With cookie httponly this isn't the case.
Say my cookie value is aes256 encrypted, base64 encoded and jwt is rsa/sha256 or other encryption encrypted.
This encryption is there so someone embedding the site in an iframe can't view and change the values.
Sure you can set X-FRAME-OPTIONS. But this doesn't protect against a custom browser that doesn't respect those headers.

Then there is the aspect of a jwt token only working in 1 browser window if using sessionStorage. And you can't have a "remember me" feature you need localStorage for that. And you need to implement your own invalidation of a token feature.

What this allows however is to not be restricted to one domain or subdomain as with cookies.

But back to the security aspect.
This is actually less secure.
Anyone embedding a target website where the user needs to log in is able to copy the token from sessionStorage and make requests as if he was the user. Of course a user would first need to log in. And like I wrote this isn't the case with a httponly cookie.

20 ^ | v • Share ›

Joe Strommen • 2 years ago

The "Stateless" and "Performance" benefits are orthogonal to Cookie vs. Header - there's no reason you can't use a cookie to store a full session object rather than an ID, and similarly you can use an auth header to pass a session ID rather than a full object.

Also, it is easy to use a CDN for your static assets with the Cookie approach - in fact one of the biggest wins of using a CDN is a separate domain so that (potentially large) cookies are not sent for every request of a static asset!

Another thing to consider is that when using auth headers, if you get XSS'd then the attacker gains access to your auth token. Not the case when using cookies (assuming you've set them as HttpOnly).

19 ^ | v • Share ›

José Romaniello Mod ↗ Joe Strommen • 2 years ago

1- Yes you can use cookie-sessions, but as shown in the image we are comparing "TRADITIONAL cookie base authentication" notice the find and deserialize session.

2- You can't serve everything from the cdn in that model. At the very least, your front page should have same origin than the endpoint that generates the cookie. Using token/auth header, your entire site can have a different origin, your api is in one domain while your static site could be in a different one. Cookies over CORS doesn't work for most browsers.

3- Agreed. Although based on my experience, it's easier to protect against XSS than protecting against CSRF. For CSRF you have to be aware of: set your cookies to HttpOnly and have a xsrf mechanism in place (which is not straightforward and not the default in many web frameworks). For XSS, you just need to make sure any input is sanitized, which usually most of the web frameworks provide something built-in.

4 ^ | v • Share ›

Daniel Lam ↗ José Romaniello • a year ago

I think in your explanation in point 2, you didn't say but it actually meant that: "Yes, cookie is fine with CDN. But CORS for cookie could be a problem".

1 ^ | v • Share ›

Wojciech Owczarczyk ↗ José Romaniello • 2 years ago

"Cookies over CORS doesn't work for most browsers." - Could you elaborate on that a little more?

^ | v • Share ›

José Romaniello Mod ↗ Wojciech Owczarczyk • 2 years ago

withCredentials will send the cookie, but there is no way to Set-Cookie in a CORS request

5 ^ | v • Share ›

nknj • 2 years ago

There is an error in the authInterceptor code: replace response with responseError
https://docs.angularjs.org/api...

6 ^ | v • Share ›

Matthew Donofrio • 2 years ago

Just some notes on CORS. We just did something similar to @JosePadilla; OAuth2 tokens, RESTful API and CORS and unfortunately, CORS proved to be problematic when used in this manner.

1. CORS doesn't play nicely with the Authorization header. That header is not considered a 'simple header' so a pre-flight request is required for all requests (which really stinks!) In addition, the OPTIONS call does not send the Authorization header so you need to account for that in your API.

2. The browser can cache the preflight responses but the cache is for per URL.

3. For our RESTful APIs, we send the Content-Type : 'application/json' header however, this will always trigger a pre-flight request also.

6 ^ | v • Share ›

José Romaniello Mod ➚ Matthew Donofrio • 2 years ago

Matthew, you are absolutely right.

Adding the Authorization header, or any non "simple header" (which is any header that you can modify, actually) will make the browser to preflight the request in scenarios where otherwise will not be required, like a GET or POST with content-type form-urlencoded.

You do not have to enforce security on OPTIONS calls, since as you mention the header is not sent. This seems problematic for Internet Information Server specially, so some browsers will just ignore the 401 status code.

The thing is that, with CORS there seems to be no way to authenticate an API and at the same time avoid the preflight request. Using cookies with the withCredentials method will require a preflight request too.

2 ^ | v • Share ›

Rcr Baker ➚ Matthew Donofrio • 2 years ago

We had the exact same situation with the preflight requests. We are still using this token based auth, but we had to move the API to the same domain as the frontend app. By the way, great article!! thanks for sharing it.

^ | v • Share ›

james • a year ago

Why do we need to send a Bearer in the interceptor ?

I am referring to :

config.headers.Authorization = 'Bearer ' + $window.sessionStorage.token;

5 ^ | v • Share ›

José Romaniello Mod ➚ james • a year ago

it is an standard and standard are good, but nothing else. In the same way you respond and handle a 401.

2 ^ | v • Share ›

Vegar Vikan • 2 years ago

Thanks for a very nice tutorial!

I think maybe there's a bug in your interceptor. You should handle `responseError`, not `response`:

```
myApp.factory('authInterceptor', function ($rootScope, $q, $window) {
return {
request: function (config) {

},
responseError: function (response) {
// handle requests that fail....
}
};
```

});

This is the only way that I got it to work, any way...

5 ^ | v • Share ›

José Padilla • 2 years ago

This is great stuff! Already implemented this for Django Rest Framework. I'm wondering what would be a solution for a user to invalidate a token? For example, if somebody got a hold of a user's token.

Also, any ideas of what a sane default for expiration time would be? Anything to low would result in the user being asked to log in again. But anything to long wouldn't be useful if your token was stolen. I'd like to allow users to be able to expire any token that is valid in that case.

A low expiration time might be cool to have if there was some kind of mechanism of refresh tokens.

4 ^ | v • Share ›

Matthew Donofrio ➚ José Padilla • 2 years ago

Here's how we solved this. We set our token expiration to 30 mins. When the token expires our API returns an 401 and Angular does 2 things:

1. Fires off an API request for a new token (using the refresh token). We actually do this in our NodeJS proxy and not from the client although I guess you could if you wanted to store the refresh token there (we did not)

2. At the same time, Angular queues up Ajax requests until a new, valid OAuth2 token is returned from step 1. When that request is complete, Angular loops through the queue and modifies the Authorization header to include the new token. After this, if fires off the Ajax requests.

Here's the service I used to do this. https://github.com/frio80/angu...

Hope that helps!

3 ^ | v • Share ›

José Romaniello Mod ➚ Matthew Donofrio • 2 years ago

Very nice Matthew, this is the way to go

1 ^ | v • Share ›

marcospgp ➚ Matthew Donofrio • a year ago

How does the token refreshing work? If no more credentials are required, how different can it be from the original token, and how more secure?

^ | v • Share ›

Matthew Donofrio ➚ marcospgp • a year ago

Our Access Token is short-lived (30 min). When this expires, the browser will send a request to our Node server which uses the Refresh Token (longer lived) to request a new Access Token from our OAuth2 server. When that completes, the Access Token is sent back to the browser and is used for the CORS requests.

From a security standpoint, this is beneficial because:
A) If the Access Token is stolen, it's only valid for 30 min.
B) The Refresh Token is never exposed to the client.

2 ^ | v • Share ›

marcospgp ➚ Matthew Donofrio • a year ago

Thank you! How could I integrate this with a social network login, such as facebook for example?

1 ^ | v • Share ›

Dhruv Bhatia ➚ Matthew Donofrio • a year ago

This is quite clever! Is there a standard name for this pattern, or did you guys originate it?

Share ›

Matthew Donofrio ↗ Dhruv Bhatia • a year ago

I doubt we were the first to do this. I'm sure others have used this type of pattern when faced with the same constraints as we were.

Share ›

raine ↗ Matthew Donofrio • a year ago

Hi Matthew, we are also considering doing the same thing. However, lets say that you have an endpoint which the js app can use to request new token, what do you do to make sure authorize the app request here? And how do you protect it from CSRF attack?

Share ›

Matthew Donofrio ↗ raine • a year ago

Good question. Our node server for generating a new token requires an Authorization header and a POST. In addition, our node server does not serve CORS headers. This means you cannot make CORS requests (which would allow you to add fake an Authorization header) plus you cannot use a HTML form POST since you do not have access to headers.

Share ›

Mani ↗ Matthew Donofrio • a year ago

Hi Matthew, with the approach being followed for token renewal, it is still vulnerable for the person with stolen token to get a new access token. Am I right? If not, how do you prevent the stolen token person from renewing the token.

Also would be good, if you could explain "our node server does not serve CORS header". Do you try to say all the api, and UI are running under same domain?

2 Share ›

Guest ↗ Matthew Donofrio • a year ago

Thank you very much! Very well written post :) I'll let you know if I have any more doubts or so, hope I'm not taking too much of your time!

Share ›

José Romaniello Mod ↗ José Padilla • 2 years ago

Very good points, tocayo!

I think with this approach is impossible to "invalidate" a token once generated, but a solution I'm thinking is to "blacklist" the token, this is for the time the token is valid, you save somewhere that the token is blacklisted, you can save this blacklist somewhere in redis or in mongodb, but the key concept to keep the collection small is to expire the document when the token expires. You can get the expiration date of the token when you decode the JWT is the "exp" field and is a unix since the epoch datetime.

So, your app will validate that the token is valid with the secret, not expired and not blacklisted.

About the expiration value, I think a sane value is more than one day, but it depends a lot on the application.

Refresh a token has high priority in our backlog we will add an API for that.

We are very interested in your django integration, is there already a JWT middleware for django? links?

Thanks for sharing!

2 Share ›

José Padilla ↗ José Romaniello • 2 years ago

Just released my Django REST framework JWT authentication implementation
https://github.com/GetBlimp/dj...

José Padilla ➚ José Romaniello • 2 years ago

Hola! Thanks for getting back to me. This is some great stuff. I wish more people catch up to it and start implementing and finding best security practices and more use cases. I was looking into what Firebase was doing to store their tokens. They currently use a cookie and localStorage with SJCL.

I've just implemented a REST API that can use a JWT and an access token(from OAuth2) via the Authorization Bearer header and it's completely transparent.

The JWT implementation I built is specific for Django REST Framework and I'll be publishing that in a few days, but it should be easy enough to make it generic.

2 ^ | v • Share ›

xeodou • 2 years ago

We use the token-auth before, but for now we use both cookie and token. Use the token-auth the issue we have is download file, we can not download file with the binary and write to the local disk. And we won't let everybody to download the file, only accept the login user. Download file can not be handle by front-end , Browser handle the operation is the best way , so we need cookie. This is the only problem we use token-auth.

4 ^ | v • Share ›

José Romaniello Mod ➚ xeodou • 2 years ago

Very good point, a solution will be to put the JWT in a query string, but have in mind that even if is HTTPS and the token is secure, the token can be seen from the history of the browser. For some, this is a security issue, but most websites are just fine with this, e.g. github uses it when you click the "RAW" link on a private project for instance.

Another options is having an specific token for the file download that you can use one time (or for a very short period of time). To generate this token , you call an API with the JWT.

One solution I would consider is HAWK (https://github.com/hueniverse/... "bewits". These are tokens valid for one specific link and just for GETs, and you can make then valid for a short period of time.

^ | v • Share ›

xeodou ➚ José Romaniello • 2 years ago

Yes, have different solution for issue. Also can use nginx handle the download link , Nginx have an module called `HttpSecureLinkModule` may good to do this . But in our project we think use cookie with token is the best way.

^ | v • Share ›

José Romaniello Mod • a year ago

Featured by Auth0 Blog

Hi everyone subscripted to this comment thread. We appreciate very much your interest for this technology, the questions and answers.

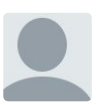
We will be closing the discussion here because it is very hard to follow and it is not easy to search.

Feel free to open specific topics regarding JWT in our discourse here:

https://ask.auth0.com/category...

Thanks again!

3 ^ | v • Share ›



abhinav • 2 years ago

I am very new to this authentication stuff. If I go on using JWT, what would be my approach to also include facebook, google, etc login in my node js website.

3 ^ | v • Share ›

fabrik42 • 2 years ago

Very good write up!

But on practice, wouldn't you want to save the token in localStorage so the user does not have to login every time he visits the site?

3 ^ | v • Share ›

John E ↗ fabrik42 • 2 years ago

This is related to the question I have. If you didn't want to have your user log in every time, is there any downside to storing token in a cookie? I am in process of hooking up this approach using golang backend.

^ | v • Share ›

José Romaniello Mod ↗ John E • 2 years ago

Yes, we mention in the article storing in session storage. The only ways the browser has to persist state is using cookies or session storage, and this is very valid.

^ | v • Share ›

Thomas • 2 years ago

I see that you are using the Authorization header to send the token on each requests. How is it better than sending it as a query string? Obviously it won't appear in the browser history but I am afraid that some proxies could mess with that header. Can we modify **cross-domain requests headers** in **IE9**?

2 ^ | v • Share ›

José Romaniello Mod ↗ Thomas • 2 years ago

Besides the history, the problem with querystring is that even in https, the url can be logged in intermediary proxies. This is not a huge problem, in fact most oauth apis accept the access_token both in header and in querystring.

You are right about "XDomainRequest" (ie8/ie9), it can't modify any headers, more info here http://blogs.msdn.com/b/ieinte....

^ | v • Share ›

Marçal Juan Llaó • 2 years ago

One issue to note, if you have your API in one domain and access the API from another subdomain you will send requests to API with the correct cookie session, but if you're using tokens the subdomain wouldn't have any chance to reuse the same token. It's this correct?

It's the main issue that blocks me the migration from cookie based auth to token.

Seems the same scenario for a widgets in another websites, how you can detect the user's login status?

Very great post! Thanks!!

2 ^ | v • Share ›

José Romaniello Mod ↗ Marçal Juan Llaó • 2 years ago

You can validate the token with the same secret in both backends. There is also another way, auth0 supports "delegation tokens", this means your application A can generate a token for B given a token for A, then B can validate the token with its own secret. You send the token in the CORS request as shown here.

Now, I'm not sure if the scenario you have is "user logins in app1.boo.com and then she goes to app2.boo.com". In this scenario you can't share the token with the local storage because of the same origin policy but if you use Auth0, your user can click on a login button on app2.boo.com and he will see "Last time you signed in with ... " and the account he signed in on app1.boo.com, no matter what identity provider he use, he will not have to enter his credentials again.

I hope this help, feel free to come to chat.auth0.com to discuss your scenarios, we really like to help and also to learn.

1 ^ | v • Share ›

Marçal Juan Llaó ↗ José Romaniello • 2 years ago

The scenario I was talking is: You access to foo.com and there's a login form that use CORs to login to app.foo.com. So the token will be stored on foo.com.
If you redirect to app.foo.com you won't have access to the token in localStorage/sessionStorage due to same origin policy, right.
I always have one API, so all authentication is to app.foo.com, there's no authenticated API hosted on foo.com.

About delegation tokens, it can apply to this scenario?, there's some resources/documentation to know deeply? Thanks!!

I'm looking forward to talk about this, very interesting subject. I have not found you in the chat, maybe next week can coincide.

Share ›

José Romaniello Mod ➚ Marçal Juan Llaó • 2 years ago

One solution for this will be to keep your authentication api in another subdomain, let's say login.foo.com. foo.com can POST the credentials to login.foo.com this will authenticate the user, store the state in a cookie and redirect back to app.foo.com with the jwt in a hash http://app.foo.com/#token=x

Later on, if the user goes directly to http://foo.com you can do a JSONP request to login.foo.com, to see if the user is logged in then you can redirect to login.foo.com that will redirect back to app.foo.com with the token.

Is important to notice that the spirit of this article is to talk about the benefits of using an Authorization header to authenticate but the way the browser has to persist state is localStorage and cookies, JWT and the Authorization header has nothing to do with this.

If you have a native app in addition to the web app, you will store the JWT in something like sqlite for native and cookies/localstorage for angular, but the authentication in server side will be the same.

We are going to write a follow up of this articles, showing how you can solve issues like this using Auth0.

2 Share ›

Steve • a year ago

Do you guys have any idea how to implement roles into express-jwt, or should I handle these on the client side (as well...)? I want different parts of my API to be accessible to different roles, lets say admin, user, superuser.

1 Share ›

Giovanni • a year ago

I suggest to specify the run() method in app.js like this

```
angular.module('myApp', [])
.run(function($window) {
if (!$window.sessionStorage.token) {
$window.location.href = "#/login";
return;
}
})
```

otherwise before redirecting to /login page the un-authenticated page is showed to the user, that is not very good from an UX point of view

1 Share ›

catrapture • 2 years ago

Brilliant post! Thanks for clearly explaining both methods - I found it most useful.

1 Share ›

Devin • 2 years ago

Whenever I try to use jwt.sign() method, I'm hit with a 'has no method 'sign'' error. Any ideas?

1 ^ | v • Share ›

Trent Nelson ↗ Devin • 2 years ago

I had the same problem. The write-up looks like it's missing this line....

var jwt = require('jsonwebtoken'); // and the npm install jsonwebtoken

It's in the source code.

4 ^ | v • Share ›

Braulio Diez • 2 years ago

What if the user right clicks and chooses open a given link in a new navigator tab? Token will be not available for that new page on that new tab? (I'm used to token, but that's something users can find annoying).

1 ^ | v • Share ›

José Romaniello Mod ↗ Braulio Diez • 2 years ago

That's why we mention using local storage or cookie which is the mechanism the browser has to save state. We explain a little bit further in the follow up:

http://blog.auth0.com/2014/01/...

^ | v • Share ›

Braulio Diez ↗ José Romaniello • 2 years ago

Got it !, really smart approach, I like it.

^ | v • Share ›

marijang • a year ago

First thanks for blog post:D It is great:D
What about "How to handle session expiration"?

I hard to find info about that. For example let say that token will last for 2 hours. What happanes after two hours? I don't wanna ask user to enter crendetials again if he is active?

^ | v • Share ›

Guest • a year ago

Definitely loving the stateless approach. I had a hard time finding an article explaining this correctly, thanks.

^ | v • Share ›

Calebe Aires • a year ago

I have implemented it on an app the use hapijs + jsonwebtoken. Working on Cordova, sometimes the $http gets 401.

I`ve tried to implemente an "handle the case where the user is not authenticated" with something like that:

```
var user = store.get('aio-profile');
data = {
email: user.email,
password: user.password
}
$http.post(host+'login', data).success(function(res){
$window.sessionStorage.token = res.token;
});
```

is it right?

^ | v • Share ›

Michael Bleigh • a year ago

I've spent a good deal of time wrestling with cross-domain client authentication, and I've come to the conclusion that a modern, CORS-based cookie approach is superior to token. I detailed it in a blog post here: http://www.divshot.com/blog/st... but the highlights are:

1. Centralized cookie-based auth means that you never have bearer tokens (even short-lived ones) laying around.
2. CORS provides everything you need to fully decouple, even while using cookies. withCredentials rocks.
3. As others have pointed out, you can store stateless signed cookies that don't require centralized lookup.
4. With a cookie you have true "single" sign-on and "single" sign-out. Once the cookie is set, any domain can access protected resources so long as it passes CORS origin checks. Once the cookie is cleared, every domain is logged out which is a far sight better than some of the ugly stuff you have to do to clear bearer tokens out of multiple clients.

I totally agree that token auth is superior to same-domain, centralized-lookup cookie auth However, I used token auth for about a year before moving to withCredentials cookie and it's been a far superior experience from a dev and user perspective both.

And it's the authentication server that sets the cookie before redirecting back to the client, so there's no issues with CORS and Set-Cookie.

▲ | ▼ • Share ›

liangyuxin.02@gmail.com • a year ago

I think this login design has hack bug.Some one can intercept user's http request and get the token.So that the hacker can simulate a request to login.Right?

▲ | ▼ • Share ›

José Romaniello Mod ➚ liangyuxin.02@gmail.com • a year ago

In the same way that someone can intercept a cookie. Use SSL.

▲ | ▼ • Share ›

Archimedes Trajano • a year ago

I am just wondering if it is a good idea to just store the id_token as a browser cookie to use that as the authentication for the rest of the REST API calls, somewhat akin to the BASIC authorization where the user ID and password is sent over and over again.

That way the Angular app or any other web app does not need to deal with auth semantics.

▲ | ▼ • Share ›

István Pató • a year ago

This application is a proof-of-concept (PoC) of using AngularJS with secured REST service with session handling and authorization. It is using JTW - JSON Web Token.

Session data hashed on the server side and immutable by the client application (but readable).

All request update the Authorization header, so token will be short-term (improve security - session management).

https://github.com/vanioinform...

▲ | ▼ • Share ›

james • a year ago

I am receiving an "Unauthorized" alert message on clicking "Sssh this is private" after logging in. What could be the reason for that ?

▲ | ▼ • Share ›

Alberto Miguel Pose Mod ➚ james • a year ago

Hi James,

There was an issue with the versions of express-jwt and jsonwebtoken dependencies. If you can re-check now it should be working.

Thanks in advance,

Share ›

anvarik • 2 years ago

Does the expiresInMinutes option work? I tried with several values, but I could not see any effect of it, at the end user is always logged in and nothing changes

Share ›

José Romaniello Mod ↗ anvarik • 2 years ago

Yes, but there was an issue that was fixed few days ago, please try with latest version.

Share ›

anvarik ↗ José Romaniello • 2 years ago

thanks, just saw in Github as well

Share ›

LocationIO • 2 years ago

Great post! One small niggle, the server sends a 401 response without a WWW-Authenticate header field. This does not appear to be compliant with http://www.w3.org/Protocols/rf...

The response MUST include a WWW-Authenticate header field (section 14.47) containing a challenge applicable to the requested resource

This makes things challenging for Android volley users:
https://stackoverflow.com/ques...

Share ›

Honza Peša • 2 years ago

Everything works quite well except I am not able to force the Express server to send me a 401 response on accessing restricted API unauthorized. :(Node console just prints "UnauthorizedError: No Authorization header was found" and that is it. On client the promise gets fulfilled with status 0. I have an HTTP interceptor ready to catch 401 (etc.) codes to buffer them and show login modal, but this cannnot get fired now. :(

Any ideas? This is how the server looks like: https://gist.github.com/smajl/...

Share ›

Honza Peša ↗ Honza Peša • 2 years ago

OK, so seems like the problem was in the middleware execution order. This works:

https://gist.github.com/smajl/...

Share ›

jeenson • 2 years ago

it says

// We are going to protect /api routes with JWT
app.use('/api', expressJwt({secret: secret}));

Is it possible to make this dynamic?. I might want certain URLS to be secure now and may later change it , or make it secure for certain users etc.
Is it possible to make it property driven or something like that?

Share ›

Matias Woloski ↗ jeenson • 2 years ago

You can add middlewares, before or after expressJwt to do whatever you want

// this is code not tested

// make api/foo public

```
app.use('/api/foo', function(req, res, next) {
req.user = { anonymous: true }; next();
});

app.use('/api/bar', expressJwt(..), is_in_role('admin'));

function is_in_role(role) {
return function(req, res, next) {
if (req.user.role !== role) return res.send(401);
next();
}
}
```

Matias

^ | v • Share ›

Peter Peerdeman • 2 years ago

I love this blog, thank you so much for writing it. Clear, concise and it works like a charm. Small sidenote: maybe include a notice that you need to install jsonwebtoken dependency to get the authenticate route code to work

^ | v • Share ›

Alberto Miguel Pose Mod ➚ Peter Peerdeman • 2 years ago

I'm glad that you like it. You are welcome!

Yes, you are right. I've added it. Thanks!

^ | v • Share ›

Nizar Noorani • 2 years ago

Quick question: Is stuff stored inside HTML session storage secure against CSRF attacks? I was under the impression that session storage can be read by JS from another domain?

^ | v • Share ›

Matias Woloski ➚ Nizar Noorani • 2 years ago

I think you mean XSS (not CSRF). Yes, that's the main thing you have to worry about when using this approach. Read more here:

http://blog.auth0.com/2014/01/...

^ | v • Share ›

Nizar Noorani ➚ Matias Woloski • 2 years ago

XSS, XSRF, Token, Cookies, Secure-Cookies - it all gets so confusing! :) Thanks for clarifying.

^ | v • Share ›

acrmuui • 2 years ago

Thanks for a very nice writeup, I really like the token based approach.

One question though: I need to check if a user contacting my API with a token has permission to edit his own profile, so I think I need to attach a userid to the token, that is not visible in the front end.

If I only base64 encode the user info and attach it to the token, im thinking the user can modify the userid, and send an PUT request and the server would still accept the valid token with the modified userid, resulting in modifying the wrong user.

Is there some way to attach encrypted data to the token using one of the available algorithms in the jsonwebtoken module, so the userid can be send encrypted, and then decrypted serverside.

^ | v • Share ›

José Romaniello Mod ➚ acrmuui • 2 years ago

The JWT is digital signed, so even if he can decode the profile part, change it and encode it again, the

signature will be not valid. He can't generate a valid signature because he doesn't know the secret.

I hope this make it more clear.

Share ›

acrmuui ➚ José Romaniello • 2 years ago

Thank you very much for answering, that makes perfect sense. I have now implemented your solution from the article and everything is working great. One last question: when using the jsonwebtoken and the express-jwt module, are the tokens stored in server memory?

Share ›

José Romaniello Mod ➚ acrmuui • 2 years ago

The tokens doesn't need to be stored anywhere, this is an stateless solution.

The jsonwebtoken module is used to generate tokens, but if you know the secret you can generate these token anywhere, not necessarily in the same web application.

express-jwt validates that the signature is valid for the given secret and that the token is not expired.

Share ›

Saransh Mohapatra • 2 years ago

How will I be sure that the front-end client sending request to the server for authentication is our front-end code or some other third-party or malicious code. I am asking this since then it is possible for other malicious codes to run and ask the user for username and password and then get the JWT from the server and thus have access to all the resources of that user.

Share ›

José Romaniello Mod ➚ Saransh Mohapatra • 2 years ago

To add something to this discussion, phishing is a serious problem regardless of token-based or cookies.

This is when a malicious site imitates the look and feel of serious websites like google and steal user credentials. You can be worried about the JWT in the same sense than the cookie, but you should be more worried that the malicious site now has the user credentials and it can save to its own database.

http://es.wikipedia.org/wiki/P...

Share ›

Matias Woloski ➚ Saransh Mohapatra • 2 years ago

First, you have to use HTTPS (SSL) so no one can sniff the traffic and get a token. That also applies to cookies. But the thing that you will miss in tokens is the "HttpOnly" flag, there is no such thing with local storage. So you have to be extra careful with XSS and sanitize your input/output.

http://blog.auth0.com/2014/01/...

Share ›

Arjun RP • 2 years ago

How to perform cross domain ajax call, when ever i try browser is throwing the error 'No 'Access-Control-Allow-Origin' header is present on the requested resource', i tried adding the header, but still its not working

Share ›

Guest • 2 years ago

How can we authenticate socket io using this mechanism?

Share ›

José Romaniello Mod ➚ Guest • 2 years ago

We wrote about this here http://blog.auth0.com/2014/01/...

1 Share ›

Rob McDiarmid • 2 years ago

Great post! Just one question. Are you missing a reference to the expressJwt middleware in the example for receiving a get to "/api/restricted"?

Share ›

José Romaniello Mod ➚ Rob McDiarmid • 2 years ago

Sorry for the delay, no, since we are declaring the middleware with app.use('/api',...), it affects all routes starting with /api.

More info here: http://expressjs.com/api.html#...

Share ›

Rob McDiarmid ➚ José Romaniello • 2 years ago

Thanks for clearing that up. Didn't know you could do that.

Share ›

Bernhard Hörmann • 2 years ago

But how to handle auth protected ressources like images without cookies? I just can't add a auth token to the header of an <img src="x"> or <img ng-src="x"> request, or?

Share ›

José Romaniello Mod ➚ Bernhard Hörmann • 2 years ago

Also in the last blog post, get a signed request:

http://blog.auth0.com/2014/01/...

1 Share ›

Justin Jackson • 2 years ago

Hi all, I am trying to roll my own so to speak with a single page app using jquery (although I am only using a few features of jquery.. ajax, document loaded, and might use response watching to check for header in response to resend). I read every post here, but I am still a little unclear how to best go about this. In my case I am deploying using WAR to Jetty. I have a single index.jsp page and some JS code (jquery, bootstrap js and my own js code). On the server side is Jersey (ajax) "framework" api for my single page app.

Couple issues I can't quite figure out. In a single page app how do you handle the UI when a user is not logged in.. (for example some pages to talk about what the app does, entice people to sign up, register pages, etc), and then after a user logs in..change the single page UI to show the logged in state of things? Mind you.. only using jquery and my vanilla js code.

Also, I'd like to try to understand using my rest api (as a sort of mvc layer/framework for ajax requests within my SPA) with user authentication. I was trying to be "stateless" with my rest api, by creating a token (using a salted hash algorithm) then storing that as the key in a Reddis DB and the user object (minimal properties) as the value. I return the hash as part of the response header (using my own custom header). In my SPA, besides trying to figure out how to use that response to redraw the UI as if the user is now logged in (so user sees new menus, different links, logout, etc), I want to ensure that subsequent requests ALL pass the token on every request. There is a way to do this with jquery where I set up a watch on the response, when the header is there, I store that value (the

see more

Share ›

Alberto Miguel Pose Mod ➚ Justin Jackson • 2 years ago

Hi Justin,

Bear in mind that you can still use token based authentication (I would recommend using a JWT library for generating the token). As you described, use your cookies as a way of storing persistent data locally.

Store the token on them and parse it back each time the page is reloaded. That should do the trick and you won't need to rely on html5 features.

Hope that helps!

Share ›

Guest • 2 years ago

Running the code from your github I get:

TypeError: Cannot read property 'email' of undefined - auth.server.js:44:33

Share ›

Matias Woloski ↗ Guest • 2 years ago

Did you follow the exact instruction from the GitHub readme? I just cloned it from scratch, run npm install, node auth.server.js opened localhost:8080 and it worked fine.
The error you are getting means one of these:
* The token is not being sent in the header
* The middleware express-jwt is misconfigured

Matias

Share ›

Robin Orheden • 2 years ago

Really good article! Would love to get your feedback/comments on the way we've implemented token based access for UserApp with Angular. https://github.com/userapp-io/...

Share ›

liuxiaolei • 2 years ago

Great Article, Thanks for sharing! I am expecting your next article on how to handle social authentication

Share ›

Chris • 2 years ago

Awesome article, and it's exactly the direction I am going in for my REST API, one question though.

How can you securely store the auth token across browser sessions? (Avoiding that the user has to login when he/she restarts the browser)

Tnx,

Chris

Share ›

David Caplan ↗ Chris • 2 years ago

Store it in localStorage rather than sessionStorage

Share ›

Hugo • 2 years ago

Great post, thank you very much!! But the font size of your blog is sooo small.. hard to read

Share ›

beernuts ↗ Hugo • 2 years ago

You might try the keyboard combo CONTROL and the PLUS key on your numeric keypad. This increases the zoom level on all browsers.

Share ›

José Romaniello Mod ↗ beernuts • 2 years ago

What browser do you guys use?

Share ›

PRODUCT

Pricing (/pricing)

Why Auth0 (/why-auth0)

How It Works (/how-it-works)

COMPANY

About Us (/about)

Blog (/blog)

Jobs (/jobs)

SECURITY

Availability & Trust (/availability-trust)

Security (/security)

White Hat (/whitehat)

LEARN

Help & Support (/support)

Documentation (https://auth0.com/docs)

Open Source (/opensource)

EXTEND

Lock (/lock)

Wordpress (/wordpress)

API Explorer (https://auth0.com/docs/apiv2)

CONTACT

Email Us (mailto:support@auth0.com)

10777 Main Street
Suite 204
Bellevue, WA 98004

SALES

+1 (888) 235-2699 (tel:+18882352699)

+1 (425) 312-6521 (tel:+14253126521)

SUPPORT

+1 (425) 559-9554 (tel:+14255599554)

Privacy Policy (/privacy) Terms of Service (/terms)