What are the main differences between JWT and OAuth authentication?

Asked 4 years, 3 months ago Active 4 months ago Viewed 298k times

409

180

I have a new SPA with a stateless authentication model using JWT. I am often asked to refer OAuth for authentication flows like asking me to send 'Bearer tokens' for every request instead of a simple token header but I do think that OAuth is a lot more complex than a simple JWT based authentication. What are the main differences, should I make the JWT authentication behave like OAuth?

I am also using the JWT as my XSRF-TOKEN to prevent XSRF but I am being asked to keep them separate? Should I keep them separate? Any help here will be appreciated and might lead to a set of guidelines for the community.

authentication oauth oauth-2.0 jwt

edited Jul 14 '20 at 18:56
Ak47
5,038 3 9 27

asked Oct 7 '16 at 4:30
Venkatesh Laguduva
10.4k 5 25 30

8 Answers

Active Oldest Votes

382

TL;DR If you have very simple scenarios, like a single client application, a single API then it might not pay off to go OAuth 2.0, on the other hand, lots of different clients (browser-based, native mobile, server-side, etc) then sticking to OAuth 2.0 rules might make it more manageable than trying to roll your own system.

As stated in another answer, JWT (Learn JSON Web Tokens) is just a token format, it defines a compact and self-contained mechanism for transmitting data between parties in a way that can be verified and trusted because it is digitally signed. Additionally, the encoding rules of a JWT also make these tokens very easy to use within the context of HTTP.

Being self-contained (the actual token contains information about a given subject) they are also a good choice for implementing stateless authentication mechanisms (aka *Look mum, no sessions!*). When going this route and the only thing a party must present to be granted access to a protected resource is the token itself, the token in question can be called a bearer token.

In practice, what you're doing can already be classified as based on bearer tokens. However, do consider that you're not using bearer tokens as specified by the OAuth 2.0 related specs (see RFC 6750). That would imply, relying on the `Authorization` HTTP header and using the `Bearer` authentication scheme.

Regarding the use of the JWT to prevent CSRF without knowing exact details it's difficult to ascertain the validity of that practice, but to be honest it does not seem correct and/or worthwhile. The following article (Cookies vs Tokens: The Definitive Guide) may be a useful read on this subject, particularly the *XSS and XSRF Protection* section.

One final piece of advice, even if you don't need to go full OAuth 2.0, I **would strongly recommend on passing your access token within the `Authorization` header instead of going with custom headers**. If they are really bearer tokens, follow the rules of RFC 6750. If not, you can always create a custom authentication scheme and still use that header.

> Authorization headers are recognized and specially treated by HTTP proxies and servers. Thus, the usage of such headers for sending access tokens to resource servers reduces the likelihood of leakage or unintended storage of authenticated requests in general, and especially Authorization headers.

(source: RFC 6819, section 5.4.1)

edited Jul 14 '20 at 18:55
Ak47
5,038 3 9 27

answered Oct 7 '16 at 9:33
João Angelo
50.5k 11 126 138

4 Does this mean if I use JWT authentication on a mobile app, I don't need to include CSRF on its POST request? Unlike web interface with forms? – user805981 Oct 12 '17 at 22:55

4 Cookies vs Tokens: The Definitive Guide , i.e auth0.com/blog/cookies-vs-tokens-definitive-guide isnt working This is another great discussion post : stackoverflow.com/questions/37582444/… – Siddharth Jain Jun 22 '18 at 3:41

2 " they are also a good choice for implementing stateless authentication mechanisms (aka Look mum, no sessions!). " If you need a way to invalidate the token because let's say it was leaked or intercepted or the user simply logged out and removing the token is not secure enough because the token is still valid then you need to store them in some database, so I think there must be some notion of session on the server for security purposes or simple token blacklist. You might say use "refresh" tokens for this. But refresh tokens can be intercepted too and consequences are much worse. – Konrad Sep 5 '18 at 7:27

2 @Konrad, I did implement a similar mechanism which stored the unused valid tokens in a table, release them from there when they expire. For each incoming request, I have written code to cross check the incoming token against the "unused valid tokens". Even though it works, I always had my doubts - there should be a better way to handle unused but still valid tokens. – Venkatesh Laguduva Sep 11 '18 at 2:22

3 On the other hand refresh tokens just complicate implementation of the client. Because if your access token expires you need to handle that, user will be pissed if you will just log him out without any possibility of even manual refresh of the session(like banks do). It's slightly more work to do, also using standard ways of authentication (OID) and authorization(OAuth) can very often be an overkill. – Konrad Sep 11 '18 at 7:19

335

OAuth 2.0 defines a protocol, i.e. specifies how tokens are transferred, JWT defines a token format.

OAuth 2.0 and "JWT authentication" have similar appearance when it comes to the (2nd) stage where the Client presents the token to the Resource Server: the token is passed in a header.

But "JWT authentication" is not a standard and does not specify *how* the Client obtains the token in the first place (the 1st stage). That is where the perceived complexity of OAuth comes from: it also defines various ways in which the Client can *obtain* an access token from something that is called an Authorization Server.

So the real difference is that JWT is just a token format, OAuth 2.0 is a protocol (that *may* use a JWT as a token format).

edited Aug 5 '19 at 9:21
piet.t
10.9k 7 39 49

answered Oct 7 '16 at 7:12
Hans Z.
38.8k 9 78 103

14 Do oAuth protocol implementations use JWT as the token format, for most cases? If not what is most common? – James Wierzba Nov 10 '17 at 23:30

20 The token format in oauth is undefined, but JWT should work fine – vikingsteve Nov 21 '17 at 14:34

151

Firstly, we have to differentiate JWT and OAuth. Basically, JWT is a token format. OAuth is an authorization protocol that can use JWT as a token. OAuth uses server-side and client-side storage. If you want to do real logout you must go with OAuth2. Authentication with JWT token can not logout actually. Because you don't have an Authentication Server that keeps track of tokens. If you want to provide an API to 3rd party clients, you must use OAuth2 also. OAuth2 is very flexible. JWT implementation is very easy and does not take long to implement. If your application needs this sort of flexibility, you should go with OAuth2. But if you don't need this use-case scenario, implementing OAuth2 is a waste of time.

XSRF token is always sent to the client in every response header. It does not matter if a CSRF token is sent in a JWT token or not, because the CSRF token is secured with itself. Therefore sending CSRF token in JWT is unnecessary.

edited Nov 1 '18 at 18:07

answered Oct 7 '16 at 6:05
Melikşah Şimşek
1,873 1 6 11

12 I don't understand why this answer has a lot of upvotes, it states that "OAuth is an authentication framework" and this is completely wrong. OAuth is an authorization protocol and only an authorization protocol. – Michael Oct 30 '18 at 16:14

5 Hi @Michael there is too much misunderstanding about this. I edited my comment thank you. – Melikşah Şimşek Nov 1 '18 at 18:08

2 Are you guys saying that Oauth is just a piece of Standards that developers should follow? Or it really is a framework? – StormTrooper Apr 27 '20 at 12:10

"If you want to do real logout you must go with OAuth2" -- Not true. For example, Devise-JWT is a non-

OAuth solution that provides "log out" functionality by invalidating tokens: github.com/waiting-for-dev/devise-jwt#revocation-strategies – GoBusto Sep 2 '20 at 11:49

1 @Michael, that's not entirely correct. The RFC is titled "The OAuth 2.0 Authorization Framework" so I guess that leaves some confusion as well ;) tools.ietf.org/html/rfc6749. I get what you mean though. – hfossli Sep 10 '20 at 7:05

80

JWT (JSON Web Tokens)- It is just a token format. JWT tokens are JSON encoded data structures contains information about issuer, subject (claims), expiration time etc. It is signed for tamper proof and authenticity and it can be encrypted to protect the token information using symmetric or asymmetric approach. JWT is simpler than SAML 1.1/2.0 and supported by all devices and it is more powerful than SWT(Simple Web Token).

OAuth2 - OAuth2 solve a problem that user wants to access the data using client software like browse based web apps, native mobile apps or desktop apps. OAuth2 is just for authorization, client software can be authorized to access the resources on-behalf of end user using access token.

OpenID Connect - OpenID Connect builds on top of OAuth2 and add authentication. OpenID Connect add some constraint to OAuth2 like UserInfo Endpoint, ID Token, discovery and dynamic registration of OpenID Connect providers and session management. JWT is the mandatory format for the token.

CSRF protection - You don't need implement the CSRF protection if you do not store token in the browser's cookie.

edited Aug 29 '20 at 2:43
Machavity ♦
28.1k 17 72 89

answered Oct 8 '17 at 0:33
ManishSingh
1,199 8 10

7 No cookies == No CSRF protection. If you don't use cookies for authorization, then you don't have to worry about CSRF protection. – niranjan harpale Oct 18 '19 at 11:17

64

It looks like everybody who answered here missed the moot point of OAUTH

From Wikipedia

> OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.[1] This mechanism is used by companies such as Google, Facebook, Microsoft and Twitter to permit the users to share information about their accounts with third party applications or websites.

The key point here is `access delegation` . Why would anyone create OAUTH when there is an id/pwd based authentication, backed by multifactored auth like OTPs and further can be secured by JWTs which are used to secure the access to the paths (like scopes in OAUTH) and set the expiry of the access

There's no point of using OAUTH if consumers access their resources(your end points) only through their trusted websites(or apps) which are your again hosted on your end points

You can go OAUTH authentication only **if you are an `OAUTH provider` in the cases where the resource owners (users) want to access their(your) resources (end-points) via a third-party client(external app).** And it is exactly created for the same purpose though you can abuse it in general

Another important note:
You're freely using the word `authentication` for JWT and OAUTH but neither provide the authentication mechanism. Yes one is a token mechanism and the other is protocol but once authenticated they are only used for authorization (access management). You've to back OAUTH either with OPENID type authentication or your own client credentials

answered Jul 16 '17 at 16:19
manikawnth
1,691 14 29

6 OAuth can also be used for your own clients, not necessarily just 3rd party ones. The Password Credentials Grant type does exactly that. – harpratap Jan 12 '18 at 2:57

3 I was looking for google for such a concrete answer but could not find one. Everyone was just talking about definitions e.g. token vs protocol. Your answer explained the true purpose of using one above the other. Thank you so much! – Vivek Goel Jan 7 '20 at 18:39

10

find the main differences between JWT & OAuth

1. OAuth 2.0 defines a protocol & JWT defines a token format.
2. OAuth can use either JWT as a token format or access token which is a bearer token.
3. OpenID connect mostly use JWT as a token format.

answered May 13 '19 at 6:38
Suraj Kumar Pandey
101 1 3

7

JWT is an open standard that defines a compact and self-contained way for securely transmitting information between parties. It is an authentication protocol where we allow encoded claims (tokens) to be transferred between two parties (client and server) and the token is issued upon the identification of a client. With each subsequent request we send the token.

Whereas OAuth2 is an authorization framework, where it has a general procedures and setups defined by the framework. JWT can be used as a mechanism inside OAuth2.

You can read more on this here

[OAuth or JWT? Which one to use and why?]

edited Mar 21 '19 at 11:00

answered Feb 21 '18 at 3:08
samuelj90
6,054 1 33 38

2

Jwt is a strict set of instructions for the issuing and validating of signed access tokens. The tokens contain claims that are used by an app to limit access to a user

OAuth2 on the other hand is not a protocol, its a delegated authorization framework. think very detailed guideline, for letting users and applications authorize specific permissions to other applications in both private and public settings. OpenID Connect which sits on top of OAUTH2 gives you Authentication and Authorization.it details how multiple different roles, users in your system, server side apps like an API, and clients such as websites or native mobile apps, can authenticate with each othe

> **Note** oauth2 can work with jwt , flexible implementation, extandable to different applications

answered Feb 4 '18 at 17:31
naila naseem
447 2 9 19

1 It appears you have this exactly backwards. – jbruni Jan 8 '20 at 17:31