Why is passing a token for each API request more secure than passing a username and secret for each API request?

Asked 10 months ago Active 10 months ago Viewed 196 times

I guess this could also be phrased as JWT vs. Basic Auth, similar idea?

0

Passing a token for each API request seems to be the more common and recommended approach, and I have the gut feeling that it just "feels better and safer", but I am not sure why and how to clearly communicate it.

authentication jwt

edited May 10 '19 at 17:21

asked May 10 '19 at 16:19
atkayla
111 2

Is the secret a password? Is API the kind used to allow a trusted 3rd party to access a user's account and do things on behalf of that person? Or is it the kind with an API key used to enforce access rights, where

Both username and secret look like 1234567-abcd-a1b2-c3d4-e5f6g7h8i9j0. Yes, API is the kind to allow 3rd party to access a user's account on behalf of that person. No, not using API Key like developers.google.com/maps/documentation/javascript/get-api-key. – atkayla May 10 '19 at 17:20

3 Answers

In most use cases, a token is a form of identification, similar to a username without a password.

If you are certain that nefarious actors or incompetence won't compromise that identification, then many people will use that as a form of authentication, such as with session tokens inside of browser cookies, or as the token in JWT.

A token is typically generated from a much higher entropy source than a username or a password, and is unlikely to be reused or shared with third parties, unlike passwords.

Additionally, if a token is compromised, it only affects one account in one system. If a user's password were compromised, it might affect most of the user's accounts across many systems, possibly also giving access to the user's bank.

(Please use a password manager and generate random passwords for every account, people!)

If the username and password are generated randomly with high entropy, and assigned only to a specific account and not re-used on other systems, then there is little difference between using a username and password, and using a token. In the end, both are simply authentication using shared secrets. Tokens just generally don't get shared with third parties the way that security-ignorant-users' passwords are.

answered May 10 '19 at 17:38

Ghedipunk

5,304 2 19 31

A username and password is a blunt instrument. They are hard to get people to change, and they represent the full authority of their bearer.

Tokens can represent authorization on a much finer scale. A username and password can be used to request a very limited token. A password might be good for 90 days or more, but a token could be issued that lasts only three minutes, making them harder to exploit. A service could issue a token good for read-only access to a specific resource, ensuring the token can't be sent to a different server for higher access.

Limited tokens also reduce a system's attack surface. The token issuing system can be secured,

passwords, it can't be used to steal or harvest them and provide elevated permissions to an attacker.

answered May 10 '19 at 18:42

John Deters

30.8k 3 48 101

0

Multiple reasons. A partial list:

Security

Per-user secrets (generally referred to as "passwords", although they needn't always be memorable) are re-usable after the session ends (even if the user doesn't re-use the password elsewhere, which users have a deplorable tendency to do), and are therefore more dangerous to the user if compromised. You cannot rely on secrets transmitted over the network or stored in the client (since they must be sent on every request) or in a server database staying secret forever.

Per-client tokens can also be scoped more tightly (for example, restricting their access to only a few functions, or to use only from a specific IP address), and can be revoked (if compromised, or even just if compromise is suspected) without invaliding all other sessions. Passwords generally need to give access to everything, from everywhere, and if they are compromised you need to change them and invalidate *all* sessions.

Finally, using session tokens means users can use a third-party identity provider without exposing their credentials to your application (this is very common; for example, you can log into StackExchange using single-sign-on via Google, without ever giving StackExchange your Google credentials). Even if you don't plan to use SSO or support third-party auth at this time, building your session management on tokens rather than credentials will make it much easier to do so in the future, without requiring your users to expose their credentials on those third-party sites to you.

Performance and Scalability

For security, passwords (secrets the user chose or that are chosen for memorability) need to be passed through a computationally-expensive hash function (sometimes described as a key derivation function, such as PBKDF2) and this is slow. A site owner really wants to only do that *once* per session, and in response, issue a token that can be verified very quickly on subsequent requests.

Additionally, tokens can be made verifiable without any request to the database at all (for an example, see JWTs). This not only avoids DB queries (except when initially issuing or refreshing the token), it also means that a request can go to a different server in a cluster than the one that handled the previous request, and the user's session can nonetheless be verified without accessing a DB (or shared cache or anything like that).

answered May 11 '19 at 1:25

CBHacking

18k 2 28 40