Invalidating JSON Web Tokens

Asked 6 years, 2 months ago Active 2 months ago Viewed 195k times

406

272

For a new node.js project I'm working on, I'm thinking about switching over from a cookie based session approach (by this, I mean, storing an id to a key-value store containing user sessions in a user's browser) to a token-based session approach (no key-value store) using JSON Web Tokens (jwt).

The project is a game that utilizes socket.io - having a token-based session would be useful in such a scenario where there will be multiple communication channels in a single session (web and socket.io)

How would one provide token/session invalidation from the server using the jwt Approach?

I also wanted to understand what common (or uncommon) pitfalls/attacks I should look out for with this sort of paradigm. For example, if this paradigm is vulnerable to the same/different kinds of attacks as the session store/cookie-based approach.

So, say I have the following (adapted from this and this):

Session Store Login:

```
app.get('/login', function(request, response) {
    var user = {username: request.body.username, password: request.body.password };
    // Validate somehow
    validate(user, function(isValid, profile) {
        // Create session token
        var token= createSessionToken();

        // Add to a key-value database
        KeyValueStore.add({token: {userid: profile.id, expiresInMinutes: 60}});

        // The client should save this session token in a cookie
        response.json({sessionToken: token});
    });
}
```

Token-Based Login:

```
var jwt = require('jsonwebtoken');
app.get('/login', function(request, response) {
    var user = {username: request.body.username, password: request.body.password };
    // Validate somehow
    validate(user, function(isValid, profile) {
    });
}
```

--

A logout (or invalidate) for the Session Store approach would require an update to the KeyValueStore database with the specified token.

It seems like such a mechanism would not exist in the token-based approach since the token itself would contain the info that would normally exist in the key-value store.

javascript node.js session session-cookies jwt

edited Feb 25 '14 at 7:13

asked Feb 24 '14 at 4:02
funseiki
6,396 ● 6 ● 28 ● 50

1 If you are using the 'express-jwt' package, you can take a look at the `isRevoked` option, or attempt to replicate the same functionality. github.com/auth0/express-jwt#revoked-tokens – Signus May 30 '15 at 0:37

1 Consider using a short expiration time on the access token and using a refresh token, with a long-lived expiration, to allow for checking of the user's access status in a database (blacklisting). auth0.com/blog/… – Rohmer Aug 17 '17 at 19:58

another option would be attaching IP address in payload while generating jwt token and checking stored IP vs incoming request for the same Ip address. ex : req.connection.remoteAddress in nodeJs. There are ISP providers that do not issue static IP per customer, I think this won't be a problem unless a client reconnects to the internet. – Gihan Sandaru Jan 11 at 18:20

20 Answers

Active Oldest Votes

371

I too have been researching this question, and while none of the ideas below are complete solutions, they might help others rule out ideas, or provide further ones.

1) Simply remove the token from the client

Obviously this does nothing for server side security, but it does stop an attacker by removing the token from existence (ie. they would have to have stolen the token prior to logout).

2) Create a token blacklist

You could store the invalid tokens until their initial expiry date, and compare them against incoming requests. This seems to negate the reason for going fully token based in the first place though, as you would need to touch the database for every request. The storage size would likely be lower though, as you would only need to store tokens that were between logout & expiry time (this is a gut feeling, and is definitely dependent on context).

3) Just keep token expiry times short and rotate them often system. The problem with this method, is that it makes it impossible to keep the user logged in between closes of the client code (depending on how long you make the expiry interval).

Contingency Plans

If there ever was an emergency, or a user token was compromised, one thing you could do is allow the user to change an underlying user lookup ID with their login credentials. This would render all associated tokens invalid, as the associated user would no longer be able to be found.

I also wanted to note that it is a good idea to include the last login date with the token, so that you are able to enforce a relogin after some distant period of time.

In terms of similarities/differences with regards to attacks using tokens, this post addresses the question: https://github.com/dentarg/blog/blob/master/_posts/2014-01-07-angularjs-authentication-with-cookies-vs-token.markdown

edited Dec 28 '19 at 21:28
Dan
2,321 ● 6 ● 60 ● 114

answered Apr 15 '14 at 16:49
Matt Way
26.3k ● 10 ● 64 ● 73

3 Excellent approach. My gut would be to do a combination of all 3, and/or, request a new token after every "n" requests (as opposed to a timer). We are using redis for in-memory object storage, and we could easily use this for case #2, and then the latency would go WAY down. – Aaron Wagner Apr 18 '14 at 14:04

2 This coding horror post offers some advice: Keep session bearing cookies (or tokens) short but make it invisible to the user - which appears to be in line with #3. My own gut (perhaps because it is more traditional) is just to have the token (or a hash of it) act as a key into white-listed session database (similar to #2) – funseiki Apr 18 '14 at 19:03

7 The article is well written, and is an elaborated version of `2)` above. While it works fine, personally I don't see much difference to traditional session stores. I guess the storage requirement would be lower, but you still require a database. The biggest appeal of JWT for me was to not use a database at all for sessions. – Matt Way Jul 31 '14 at 3:18

195 A common approach for invalidating tokens when a user changes their password is to sign the token with a hash of their password. Thus if the password changes, any previous tokens automatically fail to verify. You can extend this to logout by including a last-logout-time in the user's record and using a combination of the last-logout-time and password hash to sign the token. This requires a DB lookup each time you need to verify the token signature, but presumably you're looking up the user anyway. – Travis Terry Feb 6 '15 at 1:02

4 A blacklist can be made efficient by keeping it in memory, so that the DB need only be hit to record invalidations and remove expired invalidations and only read at server launch. Under a load-balancing architecture, the in-memory blacklist can poll the DB at short intervals, like 10s, limiting the exposure of invalidated tokens. These approaches allow the server to continue authenticating requests without per-request DB accesses. – Joe Lapp May 22 '16 at 5:48

82

JWTs is simply to change the secret.

If your server creates the JWT, signs it with a secret (JWS) then sends it to the client, simply changing the secret will invalidating all existing tokens and require all users to gain a new token to authenticate as their old token suddenly becomes invalid according to the server.

It doesn't require any modifications to the actual token contents (or lookup ID).

Clearly this only works for an emergency case when you wanted all existing tokens to expire, for per token expiry one of the solutions above is required (such as short token expiry time or invalidating a stored key inside the token).

answered Jul 16 '14 at 6:36
Andy
855 ●6 ●4

9 I think this approach isn't ideal. While it works and is certainly simple, imagine a case where you're using a public key - you wouldn't want to go and recreate that key any time you want to invalidate a single token. – Signus May 30 '15 at 0:10

1 @KijanaWoodard, a public/private key pair can be used to validate the signature as effectively the secret in the RS256 algorithm. In the example shown here he mentions changing the secret to invalidate a JWT. This can be done by either a) introducing a fake pubkey that doesn't match the signature or b) generating a new pubkey. In that situation, it is less than ideal. – Signus Sep 30 '15 at 15:09

1 @Signus - gotcha. not using the public key as the secret, but others may be relying on the public key to verify the signature. – Kijana Woodard Sep 30 '15 at 20:01

8 This is very bad solution. Main reason to use JWT is it's stateless and scales. Using a dynamic secret introduces a state. If the service is clustered across multiple nodes, you would have to synchronize the secret each time new token is issued. You would have to store secrets in a database or other external service, which would be just re-inventing cookie based authentication – Tuomas Toivonen Jan 5 '17 at 13:29

5 @TuomasToivonen, but you must sign a JWT with a secret and be able to verify the JWT with that same secret. So you must store the secret on the protected resources. If the secret is compromised you must change it and distribute that change to each of your nodes. Hosting providers with clustering/scaling usually allow you to store secrets in their service to make distributing these secrets easy and reliable. – Rohmer Aug 17 '17 at 19:43

65

This is primarily a long comment supporting and building on the answer by @mattway

Given:

Some of the other proposed solutions on this page advocate hitting the datastore on every request. If you hit the main datastore to validate every authentication request, then I see less reason to use JWT instead of other established token authentication mechanisms. You've essentially made JWT stateful, instead of stateless if you go to the datastore each time.

(If your site receives a high volume of unauthorized requests, then JWT would deny them without hitting the datastore, which is helpful. There are probably other use cases like that.)

Given:

important use cases:

User's account is deleted/blocked/suspended.

User's password is changed.

User's roles or permissions are changed.

User is logged out by admin.

Any other application critical data in the JWT token is changed by the site admin.

You cannot wait for token expiration in these cases. The token invalidation must occur immediately. Also, you cannot trust the client not to keep and use a copy of the old token, whether with malicious intent or not.

Therefore: I think the answer from @matt-way, #2 TokenBlackList, would be most efficient way to add the required state to JWT based authentication.

You have a blacklist that holds these tokens until their expiration date is hit. The list of tokens will be quite small compared to the total number of users, since it only has to keep blacklisted tokens until their expiration. I'd implement by putting invalidated tokens in redis, memcached or another in-memory datastore that supports setting an expiration time on a key.

You still have to make a call to your in-memory db for every authentication request that passes initial JWT auth, but you don't have to store keys for your entire set of users in there. (Which may or may not be a big deal for a given site.)

edited Mar 24 '17 at 14:29
mpoisot
5,794 ● 4 ● 20 ● 18

answered Apr 27 '16 at 8:45
Ed J
1,736 ● 3 ● 20 ● 19

13 I don't agree with your answer. Hitting a database does not make anything stateful; storing state on your backend does. JWT was not created so that you don't have to hit the database on each request.Every major application that uses JWT is backed by a database. JWT solves a completely different problem. en.wikipedia.org/wiki/Stateless_protocol – Julian May 17 '16 at 18:06

5 @Julian can you elaborate on this a little? Which problem does JWT really solve then? – zero01alpha Oct 5 '17 at 18:02

8 @zero01alpha Authentication: This is the most common scenario for using JWT. Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token. Information Exchange: JSON Web Tokens are a good way of securely transmitting information between parties. Because JWTs can be signed you can be sure the senders are who they say they are. See jwt.io/introduction – Julian Oct 5 '17 at 18:16

5 @Julian I don't agree with your disagreement :) JWT solves the problem (for services) to have the need to access a centralized entity providing authorization information for any given client. So instead of service A and service B have to access some resource to find out if client X has or has not permissions to do something, service A and B receive a token from X that proves his/hers permissions (most commonly issued by a 3rd party). Anyway, JWT is a tool that helps avoiding a shared state between services in a system, especially when they are controlled by more than one service provider. – LIvanov Apr 19 '18 at 14:36

41

I would keep a record of the jwt version number on the user model. New jwt tokens would set their version to this.

When you validate the jwt, simply check that it has a version number equal to the users current jwt version.

Any time you want to invalidate old jwts, just bump the users jwt version number.

answered Jun 15 '14 at 23:46
DaftMonk
665 ●6 ●9

14 This is an interesting idea, the only thing is where to store the version, as part of the purpose of tokens is it being stateless and not needing to use the database. A hard coded version would make it hard to bump, and a version number in a database would negate some of the benefits of using tokens. – Stephen Smith Jul 6 '14 at 21:17

13 Presumably you're already storing a user id in your token, and then querying the database to check that the user exists / is authorized to access the api endpoint. So you aren't doing any extra db queries by comparing the jwt token version number with the one on the user. – DaftMonk Jul 7 '14 at 5:58

5 I shouldn't say presumably, because theres a lot of situations where you might use tokens with validations that don't touch the database at all. But I think in this case its hard to avoid. – DaftMonk Jul 7 '14 at 6:16

11 What if user logs in from multiple devices? Should one token be used across them all or should login invalidate all previous ones? – meeDamian Aug 11 '14 at 2:38

10 I agree with @SergioCorrea This would make JWT almost as stateful as any other token authentication mechanism. – Ed J Apr 27 '16 at 8:56

39

Haven't tried this yet, and it is uses a lot of information based on some of the other answers. The complexity here is to avoid a server side data store call per request for user information. Most of the other solutions require a db lookup per request to a user session store. That is fine in certain scenarios but this was created in an attempt to avoid such calls and make whatever required server side state to be very small. **You will end up recreating a server side session, however small to provide all the force invalidation features. But if you want to do it here is the gist:**

Goals:

- Mitigate use of a data store (state-less).
- Ability to force log out all users.
- Ability to force log out any individual at any time.
- Ability to require password re-entry after a certain amount of time.
- Ability to work with multiple clients.
- Ability to force a re-log in when a user clicks logout from a particular client. (To prevent someone "un-deleting" a client token after user walks away - see comments for additional information)

- Use short lived (<5m) access tokens paired with a longer lived (few hours) client stored refresh-token.
- Every request checks either the auth or refresh token expiration date for validity.
- When the access token expires, the client uses the refresh token to refresh the access token.
- During the refresh token check, the server checks a small blacklist of user ids - if found reject the refresh request.
- When a client doesn't have a valid(not expired) refresh or auth token the user must log back in, as all other requests will be rejected.
- On login request, check user data store for ban.
- On logout - add that user to the session blacklist so they have to log back in. You would have to store additional information to not log them out of all devices in a multi device environment but it could be done by adding a device field to the user blacklist.
- To force re-entry after x amount of time - maintain last login date in the auth token, and check it per request.
- To force log out all users - reset token hash key.

This requires you to maintain a blacklist(state) on the server, assuming the user table contains banned user information. The invalid sessions blacklist - is a list of user ids. This blacklist is only checked during a refresh token request. Entries are required to live on it as long as the refresh token TTL. Once the refresh token expires the user would be required to log back in.

Cons:

- Still required to do a data store lookup on the refresh token request.
- Invalid tokens may continue to operate for access token's TTL.

Pros:

- Provides desired functionality.
- Refresh token action is hidden from the user under normal operation.
- Only required to do a data store lookup on refresh requests instead of every request. ie 1 every 15 min instead of 1 per second.
- Minimizes server side state to a very small blacklist.

With this solution an in memory data store like reddis isn't needed, at least not for user information as you are as the server is only making a db call every 15 or so minutes. If using reddis, storing a valid/invalid session list in there would be a very fast and simpler solution. No need for a refresh token. Each auth token would have a session id and device id, they could be stored in a reddis table on creation and invalidated when appropriate. Then they would be checked on every request and rejected when invalid.

edited May 25 '17 at 18:31

answered Mar 29 '16 at 3:54

Ashtonian
3,922 ● 2 ● 15 ● 24

has hacking skills, the user has time to recover the still-valid token to authenticate as the 1st user. It seems that there is no way avoid the need to immediately invalidate tokens, without delay. – Joe Lapp May 22 '16 at 3:49

4 Or you can remove your JWT from sesion/local storage or cookie. – Kamil Kiełczewski Jul 11 '16 at 22:33

1 Thanks @Ashtonian. After doing extensive research, I abandoned JWTs. Unless you go to extraordinary lengths to secure the secret key, or unless you delegate to a secure OAuth implementation, JWTs are much more vulnerable than regular sessions. See my full report: by.jtl.xyz/2016/06/the-unspoken-vulnerability-of-jwts.html – Joe Lapp Jan 11 '17 at 1:42

2 Using a refresh token is the key to allowing blacklisting. Great explanation: auth0.com/blog/… – Rohmer Aug 17 '17 at 19:56

1 This seems to me the best answer as it combines a short-lived access token with a long-lived refresh token that can be blacklisted. On logout, client should delete the access token so a 2nd user can't get access (even though the access token will remain valid for a few more minutes after logout). @Joe Lapp says a hacker (2nd user) get the access token even after it's been deleted. How? – M3RS Dec 6 '18 at 13:19

13

An approach I've been considering is to always have an `iat` (issued at) value in the JWT. Then when a user logs out, store that timestamp on the user record. When validating the JWT just compare the `iat` to the last logged out timestamp. If the `iat` is older, then it's not valid. Yes, you have to go to the DB, but I'll always be pulling the user record anyway if the JWT is otherwise valid.

The major downside I see to this is that it'd log them out of all their sessions if they're in multiple browsers, or have a mobile client too.

This could also be a nice mechanism for invalidating all JWTs in a system. Part of the check could be against a global timestamp of the last valid `iat` time.

answered Jul 25 '15 at 6:37
Brack Mo
139 ● 1 ● 2

1 Good thought! To solve the "one device" problem is to make this a contingency feature rather than a logout. Store the a date on the user record that invalidates all tokens issued before it. Something like `token_valid_after`, or something. Awesome! – OneHoopyFrood Nov 4 '15 at 18:12

1 Hey @OneHoopyFrood you have an example code to help me understand the idea in a better way? I really appreciate your help! – alexventuraio Jan 9 '16 at 23:37

2 Like all the other proposed solutions, this one requires a database lookup which is the reason this question exists because avoiding that lookup is the most important thing here! (Performance, scalability). Under normal circumstances you don't need a DB lookup to have the user data, you already got it from the client. – Rob Evans Apr 26 '18 at 10:12

8

I'm a bit late here, but I think I have a decent solution.

I have a "last_password_change" column in my database that stores the date and time when the password was last changed. I also store the date/time of issue in the JWT. When validating a

answered Dec 25 '15 at 21:13
Matas Kairaitis
148 ● 1 ● 5

1 How do you reject the token? Can you show a brief example code? – alexventuraio Jan 9 '16 at 23:39

1 `if (jwt.issue_date < user.last_pw_change) { /* not valid, redirect to login */}` – Vanuan Jan 25 '16 at 21:07

14 Requires a db lookup! – Rob Evans Apr 26 '18 at 10:12

5

You can have a "last_key_used" field on your DB on your user's document/record.

When the user logs in with user and pass, generate a new random string, store it in the last_key_used field, and add it to the payload when signing the token.

When the user logs in using the token, check the last_key_used in the DB to match the one in the token.

Then, when user does a logout for instance, or if you want to invalidate the token, simply change that "last_key_used" field to another random value and any subsequent checks will fail, hence forcing the user to log in with user and pass again.

answered May 12 '16 at 1:32
NickVarcha
103 ● 2 ● 8

This is the solution I've been considering, but it has these drawbacks: (1) you're either doing a DB-lookup on each request to check the random (nullifying the reason for using tokens instead of sessions) or you're only checking intermittently after a refresh token has expired (preventing users from logging out immediately or sessions from being terminated immediately); and (2) logging out logs the user out from all browsers and all devices (which is not conventionally expected behavior). – Joe Lapp May 22 '16 at 3:56

You don't need to change the key when the user logs out, only when they change their password or -if you provide it- when they choose to log out from all devices – NickVarcha Jul 15 '16 at 2:05

3

Keep an in-memory list like this

```
user_id   revoke_tokens_issued_before
-------------------------------------
123       2018-07-02T15:55:33
567       2018-07-01T12:34:21
```

If your tokens expire in one week then clean or ignore the records older than that. Also keep only the most recent record of each user. The size of the list will depend on how long you keep your tokens and how often users revoke their tokens. Use db only when the table changes. Load the table in memory when your application starts.

2 Most production sites run on more than one server so this solution will not work. Adding Redis or similar interpocess cache significantly complicates the system and often brings more problems than solutions. – user2555515 Mar 21 '19 at 15:54

@user2555515 all servers can be synchronized with the database. It is your choice hitting the database everytime or not. You could tell what problems it brings. – Eduardo Apr 4 '19 at 23:42

3

------------------------Bit late for this answer but may be it will help to someone------------------------

From the Client Side, the easiest way is to remove the token from the storage of browser.

But, What if you want to destroy the token on the Node server -

The problem with JWT package is that it doesn't provide any method or way to destroy the token. You may use different methods with respect to JWT which are mentioned above. But here i go with the jwt-redis.

So in order to destroy the token on the serverside you may use **jwt-redis package instead of JWT**

This library (jwt-redis) completely repeats the entire functionality of the library jsonwebtoken, with one important addition. Jwt-redis allows you to store the token label in redis to verify validity. The absence of a token label in redis makes the token not valid. To destroy the token in jwt-redis, there is a destroy method

it works in this way :

1) **Install jwt-redis from npm**

2) **To Create -**

```
var redis = require('redis');
var JWTR =  require('jwt-redis').default;
var redisClient = redis.createClient();
var jwtr = new JWTR(redisClient);

jwtr.sign(payload, secret)
    .then((token)=>{
            // your code
    })
    .catch((error)=>{
            // error handling
    });
```

3) **To verify -**

```
jwtr.verify(token, secret);
```

4) **To Destroy -**

Note : you can provide expiresIn during signin of token in the same as it is provided in JWT.

May be this will help to someone

edited Sep 6 '19 at 13:31

answered Sep 6 '19 at 13:10
Aman Kumar Gupta
469 ●6 ●5

2

Why not just use the jti claim (nonce) and store that in a list as a user record field (db dependant, but at very least a comma-separated list is fine)? No need for separate lookup, as others have pointed out presumably you want to get the user record anyway, and this way you can have multiple valid tokens for different client instances ("logout everywhere" can reset the list to empty)

answered Mar 9 '16 at 15:01
davidkomer
2,550 ●1 ●16 ●39

Yes, this. Maybe make a one-to-many relation between the user table and a new (session) table, so you can store meta data along with the jti claims. – Peter Lada Jul 24 '19 at 8:27

2

Unique per user string, and global string hashed together

to serve as the JWT secret portion allow both individual and global token invalidation. Maximum flexibility at the cost of a db lookup/read during request auth. Also easy to cache as well, since they are seldom changing.

answered Aug 30 '16 at 15:10
Mark Essel
3,712 ●2 ●23 ●45

1 Some more detail would suffice – giantas Dec 14 '19 at 15:18

1 @giantas, i think what Mark mean is the signature part. So instead using only single key to sign the JWT, combine it a key that unique for each client. Therefore, if you want invalidate a user's all session, just change the key for that user and if you to invalidate all session in your system, just change that global single key. – Tommy Aria Pradana Apr 3 at 15:39

2

1. Give 1 day expiry time for the tokens
2. Maintain a daily blacklist.
3. Put the invalidated / logout tokens into the blacklist

For token validation, check for the token expiry time first and then the blacklist if token not expired.

Ebru Yener
535 ● 4 ● 14

4 put tokens into blacklist and there goes your statelessness – Kerem Baydoğan Mar 21 '17 at 17:37

2

Late to the party, MY two cents are given below after some research. During logout, make sure following things are happening...

Clear the client storage/session

Update the user table last login date-time and logout date-time whenever login or logout happens respectively. So login date time always should be greater than logout (Or keep logout date null if the current status is login and not yet logged out)

This is way far simple than keeping additional table of blacklist and purging regularly. Multiple device support requires additional table to keep loggedIn, logout dates with some additional details like OS-or client details.

answered Feb 4 '17 at 22:47
Shamseer
564 ● 1 ● 5 ● 21

1

I did it the following way:

1. Generate a `unique hash`, and then store it in **redis** and your **JWT**. This can be called a **session**
   - We'll also store the number of **requests** the particular **JWT** has made - Each time a jwt is sent to the server, we increment the **requests** integer. (this is optional)

So when a user logs in, a unique hash is created, stored in redis and injected into your **JWT**.

When a user tries to visit a protected endpoint, you'll grab the unique session hash from your **JWT**, query redis and see if it's a match!

We can extend from this and make our **JWT** even more secure, here's how:

Every **X** requests a particular **JWT** has made, we generate a new unique session, store it in our **JWT**, and then blacklist the previous one.

This means that the **JWT** is constantly changing and stops stale **JWT**'s being hacked, stolen, or something else.

answered Oct 26 '16 at 8:49
James111
10.6k ● 11 ● 58 ● 91

1 You can hash the token itself and store that value in redis, rather than inject a new hash into the token. –

1

If you want to be able to revoke user tokens, you can keep track of all issued tokens on your DB and check if they're valid (exist) on a session-like table. The downside is that you'll hit the DB on every request.

I haven't tried it, but i suggest the following method to allow token revocation while keeping DB hits to a minimum -

To lower the database checks rate, divide all issued JWT tokens into X groups according to some deterministic association (e.g., 10 groups by first digit of the user id).

Each JWT token will hold the group id and a timestamp created upon token creation. e.g., `{ "group_id": 1, "timestamp": 1551861473716 }`

The server will hold all group ids in memory and each group will have a timestamp that indicates when was the last log-out event of a user belonging to that group. e.g., `{ "group1": 1551861473714, "group2": 1551861487293, ... }`

Requests with a JWT token that have an older group timestamp, will be checked for validity (DB hit) and if valid, a new JWT token with a fresh timestamp will be issued for client's future use. If the token's group timestamp is newer, we trust the JWT (No DB hit).

So -

1. We only validate a JWT token using the DB if the token has an old group timestamp, while future requests won't get validated until someone in the user's group will log-out.
2. We use groups to limit the number of timestamp changes (say there's a user logging in and out like there's no tomorrow - will only affect limited number of users instead of everyone)
3. We limit the number of groups to limit the amount of timestamps held in memory
4. Invalidating a token is a breeze - just remove it from the session table and generate a new timestamp for the user's group.

answered Mar 6 '19 at 8:44
Arik
3,159 ● 1 ● 14 ● 17

Same list can be kept in memory (application for c#) and it would eliminate the need for hitting the db for each request. The list can be loaded from db on application start – dvdmn Mar 25 '19 at 14:59

1

If "logout from all devices" option is acceptable (in most cases it is):

- Add the token version field to the user record.
- Add the value in this field to the claims stored in the JWT.
- Increment the version every time the user logs out.
- When validating the token compare its version claim to the version stored in the user record

A db trip to get the user record in most cases is required anyway so this does not add much overhead to the validation process. Unlike maintaining a blacklist, where DB load is significant due to the necessity to use a join or a separate call, clean old records and so on.

answered Apr 9 '19 at 17:37
user2555515
123 ● 1 ● 7

-1

This seems really difficult to solve without a DB lookup upon every token verification. The alternative I can think of is keeping a blacklist of invalidated tokens server-side; which should be updated on a database whenever a change happens to persist the changes across restarts, by making the server check the database upon restart to load the current blacklist.

But if you keep it in the server memory (a global variable of sorts) then it's not gonna be scalable across multiple servers if you are using more than one, so in that case you can keep it on a shared Redis cache, which should be set-up to persist the data somewhere (database? filesystem?) in case it has to be restarted, and every time a new server is spun up it has to subscribe to the Redis cache.

Alternative to a black-list, using the same solution, you can do it with a hash saved in redis per session as this other answer points out (not sure that would be more efficient with many users logging in though).

Does it sound awfully complicated? it does to me!

Disclaimer: I have not used Redis.

edited Sep 2 '19 at 21:24

answered Sep 2 '19 at 21:16
Jose P. V.
718 ● 1 ● 6 ● 22

-1

If you are using axios or a similar promise-based http request lib you can simply destroy token on the front-end inside the `.then()` part. It will be launched in the response .then() part after user executes this function (result code from the server endpoint must be ok, 200). After user clicks this route while searching for data, if database field `user_enabled` is false it will trigger destroying token and user will immediately be logged-off and stopped from accessing protected routes/pages. We don't have to await for token to expire while user is permanently logged on.

```
function searchForData() {   // front-end js function, user searches for the data
    // protected route, token that is sent along http request for verification
    var validToken = 'Bearer ' + whereYouStoredToken; // token stored in the browser

    // route will trigger destroying token when user clicks and executes this func
    axios.post('/my-data', {headers: {'Authorization': validToken}})
     .then((response) => {
   // If Admin set user_enabled in the db as false, we destroy token in the browser
localStorage
       if (response.data.user_enabled === false) {  // user_enabled is field in the db
           window.localStorage.clear();  // we destroy token and other credentials
```

```
        console.log(e);
    });
}
```

edited Nov 14 '19 at 15:41

answered Nov 14 '19 at 15:29
Dan
85 ● 1 ● 8

-3

I just save token to users table, when user login I will update new token, and when auth equal to the user current jwt.

I think this is not the best solution but that work for me.

edited May 22 '18 at 15:10

answered May 22 '18 at 15:04
Vo Manh Kien
15 ● 1 ● 7

2 Of course it is not the best ! Anyone having access to the db can easily impersonate any user. – user2555515 Apr 9 '19 at 17:45

@user2555515 This solution works fine if the token that is stored on the database is encrypted, just like any password stored on the database should be. There is a difference between `Stateless JWT` and `Stateful JWT` (which is very similar to sessions). `Stateful JWT` can benefits from maintaining a token whitelist. – TheDarkIn1978 Jul 28 '19 at 4:55

Highly active question. Earn 10 reputation in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.