[express-jwt](https://www.npmjs.com/package/express-jwt)

Top of Form

Bottom of Form

Middleware that validates JsonWebTokens and sets req.user.

This module lets you authenticate HTTP requests using JWT tokens in your Node.js applications. JWTs are typically used to protect API endpoints, and are often issued using OpenID Connect.

**Install**

$ npm install express-jwt

**Usage**

The JWT authentication middleware authenticates callers using a JWT. If the token is valid, req.userwill be set with the JSON object decoded to be used by later middleware for authorization and access control.

For example,

var jwt **=** require('express-jwt');

app.get('/protected',

  jwt({secret**:** 'shhhhhhared-secret'}),

  function(req, res) {

**if** (**!**req.user.admin) **return** res.sendStatus(401);

    res.sendStatus(200);

  });

You can specify audience and/or issuer as well:

jwt({ secret**:** 'shhhhhhared-secret',

  audience**:** 'http://myapi/protected',

  issuer**:** 'http://issuer' })

If the JWT has an expiration (exp), it will be checked.

If you are using a base64 URL-encoded secret, pass a Buffer with base64 encoding as the secret instead of a string:

jwt({ secret**:** **new** Buffer('shhhhhhared-secret', 'base64') })

Optionally you can make some paths unprotected as follows:

app.use(jwt({ secret**:** 'shhhhhhared-secret'}).unless({path**:** ['/token']}));

This is especially useful when applying to multiple routes. In the example above, path can be a string, a regexp, or an array of any of those.

For more details on the .unless syntax including additional options, please see [**express-unless**](https://github.com/jfromaniello/express-unless).

This module also support tokens signed with public/private key pairs. Instead of a secret, you can specify a Buffer with the public key

var publicKey **=** fs.readFileSync('/path/to/public.pub');

jwt({ secret**:** publicKey });

By default, the decoded token is attached to req.user but can be configured with the requestProperty option.

jwt({ secret**:** publicKey, requestProperty**:** 'auth' });

The token can also be attached to the result object with the resultProperty option. This option will override any requestProperty.

jwt({ secret**:** publicKey, resultProperty**:** 'locals.user' });

Both resultProperty and requestProperty utilize **[lodash.set](https://lodash.com/docs/4.17.2" \l "set)** and will accept nested property paths.

A custom function for extracting the token from a request can be specified with the getToken option. This is useful if you need to pass the token through a query parameter or a cookie. You can throw an error in this function and it will be handled by express-jwt.

app.use(jwt({

  secret**:** 'hello world !',

  credentialsRequired**:** false,

  getToken**:** function fromHeaderOrQuerystring (req) {

**if** (req.headers.authorization **&&** req.headers.authorization.split(' ')[0] **===** 'Bearer') {

**return** req.headers.authorization.split(' ')[1];

    } **else** **if** (req.query **&&** req.query.token) {

**return** req.query.token;

    }

**return** null;

  }

}));

**Multi-tenancy**

If you are developing an application in which the secret used to sign tokens is not static, you can provide a callback function as the secret parameter. The function has the signature: function(req, payload, done):

* req (Object) - The express request object.
* payload (Object) - An object with the JWT claims.
* done (Function) - A function with signature function(err, secret) to be invoked when the secret is retrieved.
  + err (Any) - The error that occurred.
  + secret (String) - The secret to use to verify the JWT.

For example, if the secret varies based on the [**JWT issuer**](http://self-issued.info/docs/draft-ietf-oauth-json-web-token.html#issDef):

var jwt **=** require('express-jwt');

var data **=** require('./data');

var utilities **=** require('./utilities');

var secretCallback **=** function(req, payload, done){

  var issuer **=** payload.iss;

  data.getTenantByIdentifier(issuer, function(err, tenant){

**if** (err) { **return** done(err); }

**if** (**!**tenant) { **return** done(**new** Error('missing\_secret')); }

    var secret **=** utilities.decrypt(tenant.secret);

    done(null, secret);

  });

};

app.get('/protected',

  jwt({secret**:** secretCallback}),

  function(req, res) {

**if** (**!**req.user.admin) **return** res.sendStatus(401);

    res.sendStatus(200);

  });

**Revoked tokens**

It is possible that some tokens will need to be revoked so they cannot be used any longer. You can provide a function as the isRevoked option. The signature of the function is function(req, payload, done):

* req (Object) - The express request object.
* payload (Object) - An object with the JWT claims.
* done (Function) - A function with signature function(err, revoked) to be invoked once the check to see if the token is revoked or not is complete.
  + err (Any) - The error that occurred.
  + revoked (Boolean) - true if the JWT is revoked, false otherwise.

For example, if the (iss, jti) claim pair is used to identify a JWT:

var jwt **=** require('express-jwt');

var data **=** require('./data');

var utilities **=** require('./utilities');

var isRevokedCallback **=** function(req, payload, done){

  var issuer **=** payload.iss;

  var tokenId **=** payload.jti;

  data.getRevokedToken(issuer, tokenId, function(err, token){

**if** (err) { **return** done(err); }

**return** done(null, **!!**token);

  });

};

app.get('/protected',

  jwt({secret**:** 'shhhhhhared-secret',

    isRevoked**:** isRevokedCallback}),

  function(req, res) {

**if** (**!**req.user.admin) **return** res.sendStatus(401);

    res.sendStatus(200);

  });

**Error handling**

The default behavior is to throw an error when the token is invalid, so you can add your custom logic to manage unauthorized access as follows:

app.use(function (err, req, res, next) {

**if** (err.name **===** 'UnauthorizedError') {

    res.status(401).send('invalid token...');

  }

});

You might want to use this module to identify registered users while still providing access to unregistered users. You can do this by using the option *credentialsRequired*:

app.use(jwt({

  secret**:** 'hello world !',

  credentialsRequired**:** false

}));

**Related Modules**

* [**jsonwebtoken**](https://github.com/auth0/node-jsonwebtoken) — JSON Web Token sign and verification
* [**express-jwt-permissions**](https://github.com/MichielDeMey/express-jwt-permissions) - Permissions middleware for JWT tokens

**Tests**

$ npm install

$ npm test