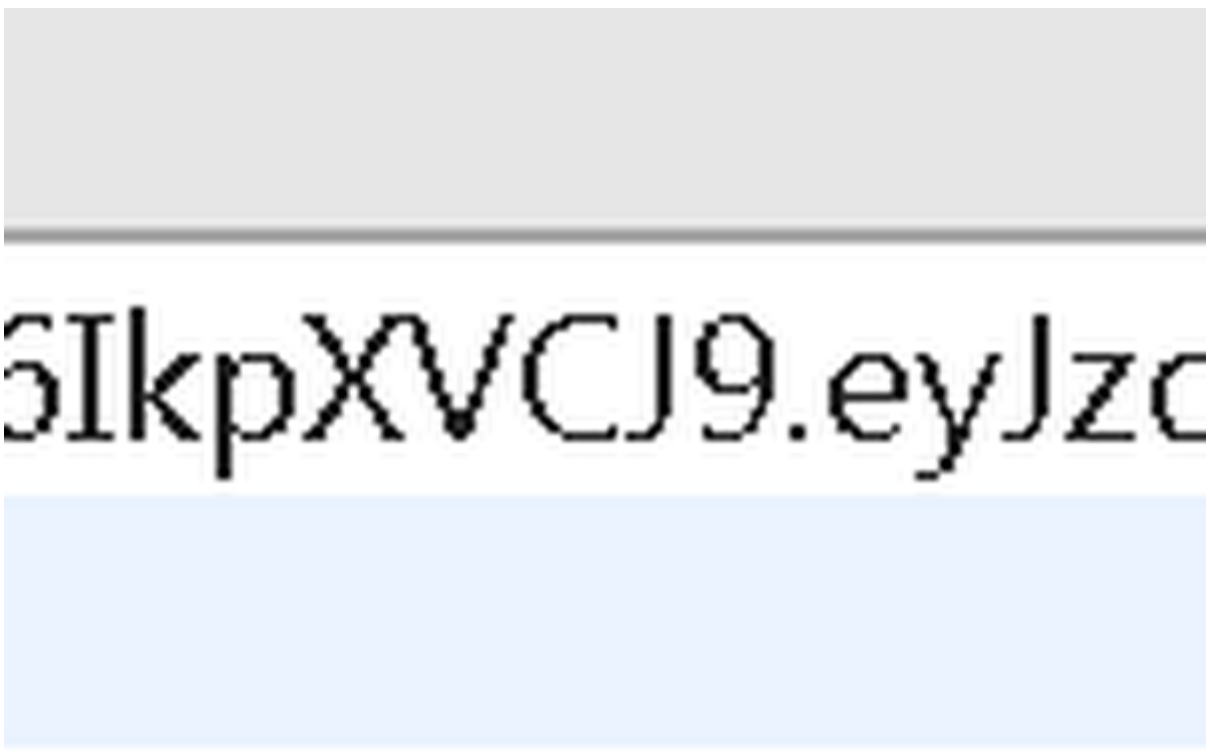
JSON web token authentication with Flask and Angularjs



JSON web tokens (JWT) are a mechanism in which a token is used instead of a username/password to authenticate API users. Token's are more secure because they can contain a scope (Access Level) and an Expiry. Thus in case of a compromise the attacker has very limited access to your data. They can also be encrypted and stored on the client side.

In this tutorial I will be describing how you can use JSON Web Tokens to authenticate API requests with Flask and Angularjs.

Note: This tutorial is based on Python 3.4 and you can use this requirements (https://github.com/Leo-g/Flask-Angularjs-JSON-Auth/blob/master/requirements.txt) file to install the required python modules via PIP.

Encoding and Decoding Tokens with PyJWT

PyJWT is a simple Python library that allows you to encode and decode web tokens. In order to create a token, PyJWT needs a JSON formatted Payload, a secret key and an optional encryption algorithm. As soon as a user is authenticated we can create the token for them as follows:

```
def create_token(user):
  payload = {
        # subject
        'sub': user.id,
        #issued at
        'iat': datetime.utcnow(),
        #expiry
        'exp': datetime.utcnow() + timedelta(days=1)
     }
   token = jwt.encode(payload, SECRET_KEY, algorithm='HS256')
   return token.decode('unicode_escape')
```

```
def parse_token(req):
    token = req.headers.get('Authorization').split()[1]
    return jwt.decode(token, SECRET_KEY, algorithms='HS256')
```

For a list of information the payload can contain see jwt.io.

Login decorator for API endpoints

The login decorator is what you can use to decorate your API resource functions or classes so that only requests with valid tokens can access them.

Example Python3 code of Login decorator

```
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        if not request.headers.get('Authorization'):
            response = jsonify(message='Missing authorization header')
            response.status_code = 401
            return response
        try:
            payload = parse_token(request)
        except DecodeError:
            response = jsonify(message='Token is invalid')
            response.status_code = 401
            return response
        except ExpiredSignature:
            response = jsonify(message='Token has expired')
            response.status_code = 401
            return response
        g.user_id = payload['sub']
        return f(*args, **kwargs)
    return decorated_function
```

I have used Flask-Restful to create my Flask API, hence I need to sub-class the Resource class and add the login decorator to its “method_decorators” property as follows:

```
class Resource(flask_restful.Resource):
    method_decorators = [login_required]
```

```
class User(Resource):
    def get(self):
                results = Users.query.all()
                users = schema.dump(results, many=True).data
                return jsonify({"users":users})
```

I added relevant code so it’s easier to understand. The complete code is available at https://github.com/Leo-g/Flask-Angularjs-JSON-Auth/blob/master/app/users/views.py (https://github.com/Leo-g/Flask-Angularjs-JSON-Auth/blob/master/app/users/views.py).

Using Angularjs module Satellizer to store and pass JWT in our API requests

Satellizer is an easy to use angularjs module for token based authentication. You will first need to configure it with the login url from where it can access the Token. Then you can use methods like $auth.login(UserCreds) (https://github.com/sahat/satellizer#authloginuser-options) , $auth.isAuthenticated() (https://github.com/sahat/satellizer#authisauthenticated) and $auth.logout() (https://github.com/sahat/satellizer#authlogout) to authenticate, verify and logout a user.

Example app.js code

```
angular.module('myApp', ['ui.router', 'ngResource', 'myApp.controllers', 'myApp.services',
angular.module('myApp').config(function($stateProvider, $urlRouterProvider, $authProvider)
  // Satellizer configuration that specifies which API
            // route the JWT should be retrieved from
            $authProvider.loginUrl = '/api/login';
```

```
 $urlRouterProvider.otherwise('/');
```

```
$stateProvider. state('login', {
url: '/login',
templateUrl: 'partials/login.html',
controller: 'LoginController',
    resolve: {
//Function to check if user is already logged in
          skipIfLoggedIn: skipIfLoggedIn
        }
```

```
 })
```

Example login.html code

```
<form>
<input required="" type="email" placeholder="Email" />
```

```
<input required="" type="password" placeholder="Password" />
```

```
<button class="button-primary">Login</button></form>
```

Example controller.js code to Login and Logout a user

```
#Relevant controller code
#complete code at  https://github.com/Leo-g/Flask-Angularjs-JSON-Auth/blob/master/angularjs
angular.module('myApp.controllers', []).controller('LoginController', function($auth, $stat
$scope.login = function() {
            $scope.credentials = {
                email: $scope.email,
                password: $scope.password
            }
```

```
          // Use Satellizer's $auth.login method to verify the username and password
            $auth.login($scope.credentials).then(function(data) {
                // If login is successful, redirect to users list
```

```
$state.go('users');
            })
            .catch(function(response){ // If login is unsuccessful, display relevant error
               toaster.pop({
                type: 'error',
                title: 'Login Error',
                body: response.data,
                showCloseButton: true,
                timeout: 0
                });
               });
        }
```

```
}).controller('LogoutCtrl', function($auth,  $location, toaster) { // Logout the user if th
```

```
if (!$auth.isAuthenticated()) { return; }
     $auth.logout()
      .then(function() {
```

```
       toaster.pop({
                type: 'success',
                body: 'Logging out' ,
                showCloseButton: true
```

```
               });
        $location.url('/');
      });
```

You will also need to configure any routes that need authentication as follows

```
$stateProvider.state('users', {
    url: '/',
    templateUrl: 'partials/users.html',
    controller: 'UserListController',
   //resolve only for authenticated users
    resolve: {
          loginRequired: loginRequired
        }
```

```
function loginRequired($q, $location, $auth, $state) {
      var deferred = $q.defer();
      if ($auth.isAuthenticated()) {
        deferred.resolve();
      } else {
        $location.path('/login');
```

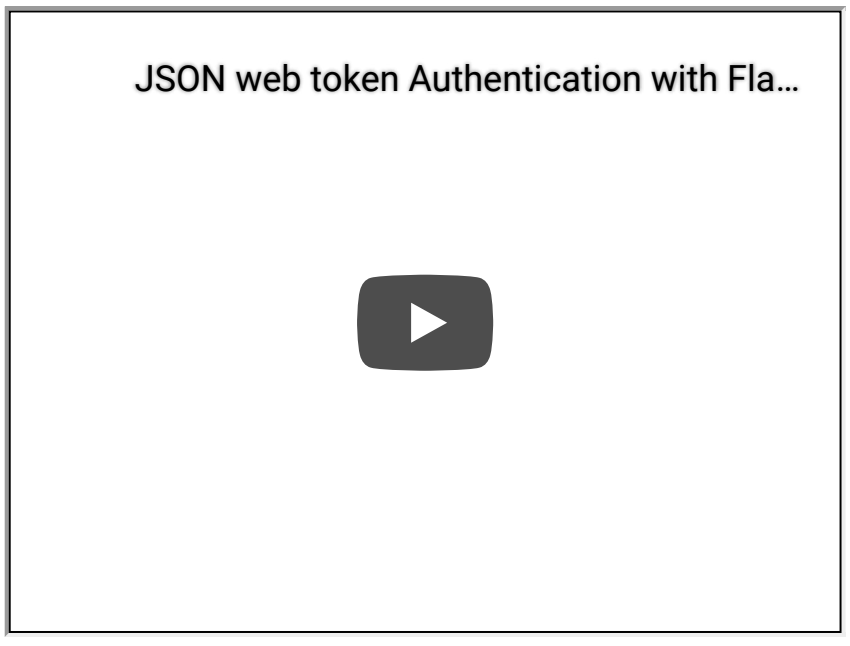
```
     }
      return deferred.promise;
```

```
   }
```

You can view the complete javascript code at https://github.com/Leo-g/Flask-Angularjs-JSON-Auth/tree/master/angularjs-frontend/js (https://github.com/Leo-g/Flask-Angularjs-JSON-Auth/tree/master/angularjs-frontend/js)

For a Live demo check https://github.com/Leo-G/Flask-Scaffold (https://github.com/Leo-G/Flask-Scaffold)

Here is a video of the working demo.



Let me know if you have any questions or suggestions in the comment section below.

Flask is an easy and simple framework for python here (/blog/a-beginners-guide-to-developing-web-applications-on-linux-with-python-3-and-flask/) is how you can get started with it.

Ref

http://flask-restful.readthedocs.org/en/latest/extending.html#resource-method-decorators (http://flask-restful.readthedocs.org/en/latest/extending.html#resource-method-decorators)

https://github.com/jpadilla/pyjwt (https://github.com/jpadilla/pyjwt)

https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage/ (https://stormpath.com/blog/where-to-store-your-jwts-cookies-vs-html5-web-storage/)

https://tools.ietf.org/html/rfc7519#page-6

(ht... (ht... (ht... (ht...

Comments

Disqus seems to be taking longer than usual. Reload?

Subscribe

Email Address *

Subscribe

Search

Search Everything Go