NodeJS JWT Authentication

We'll be creating a REST API that feature a mixture of authenticated and un-authenticated JSON endpoints and we'll be implementing a nice and simple JWT validation function that will verify incoming requests to ensure they have the appropriate authorization header set and that the value of that header is a verified JSON Web Token!

Goals

- **We will have created a really simple REST API using ExpressJS**
- **We will understand how we can create middleware functions that we can wrap endpoints with.**
- **We will understand how to add JWT Authentication to these middleware functions to ensure only authorized requestors can hit our REST API endpoints.**

Prerequisites

- **NodeJS installed on your machine**
- **A Text Editor (I recommend Visual Studio Code)**

Introduction

Being able to verify who is hitting your API endpoints is incredibly important. It allows you to determine if the requestor is an authorized person who is allowed to actually interact with your service.

This is where JWT authentication can solve our problems. We can create use JWTs in combination with request headers to help us validate that an incoming request is authorized or not.

JWTs or JSON Web Tokens

JWTs or a JSON Web Token, to use its full name, is a means of representing claims between two parties.

Structure

A typical JWT features a Header, a Payload, and a Signature.

Let’s break down what each of these is used for:

- **The Header** - This contains metadata about the JWT such as the type of token and the cryptographic algorithm used to secure it.

- **The Payload** - This is the *set of claims* contained within the JWT that contains a series of key/value pairs. The payload can be read by anyone, so ensure that you don’t add any confidential to the payload.

- **The Signature** - This is used to validate that everything in the JWT has not been tampered with in any way and is generated using the header, the payload and a signing key or passphrase.

- we are going to be creating JWTs that feature an incredibly simple payload. We’ll be then signing any JWTs we create using a private RSA key.

This private RSA key is *only* known by the server issuing any JWTs used and thus, unless a bad actor gets a hold of this key, it’s almost impossible to generate a JWT token that will have a valid signature.

A Basic REST API

Now that we have covered the theory, let’s set about creating a simple REST API in NodeJS that uses Express.JS. Create a new directory in which our project will live and run the following:

```
$ npm init
```

Keep everything as default values for now, we can edit this later should we need to.

Now that we have initialized our project, we’ll need to install the express.js Node Module:

```
$ npm install express --save
```

This will go away and fetch the required express node module which we'll be using to construct the REST API that we will be securing.

Now that we have all of our dependencies installed, we can go ahead and write some code! Create a new file within your project directory called **index.js**.

index.js

```
const express = require('express')

const app = express()

const port = 3000



app.get('/', (req, res) => res.send('Hello World!'))

app.listen(port,

  () => console.log(`Simple Express app listening on port ${port}!`))
```

This is an incredibly simple application that returns **Hello World!** whenever you hit the **/** path on port **3000**. You can subsequently start this application like so:

```
$ node index.js
```

```
Example app listening on port 3000!
```

With this now running, navigate to **http://localhost:3000/** in your browser and you will see your application returning the expected **Hello World!** back to you.

Adding a few Endpoints

Now that we have a basic NodeJS application up and running on port 3000, let’s look at how we can extend this and add a few endpoints which we can subsequently protect.

Open up your **index.js** file once again. Let’s add a few new **HTTP GET** endpoints:

index.js

```
const express = require('express')

const app = express()

const port = 3000



app.get('/', (req, res) => res.send('Hello World!'))



// let's first add a /secret api endpoint that we will be protecting

app.get('/secret', (req, res) => {

   res.json({ "message" : "THIS IS SUPER SECRET, DO NOT SHARE!" })

})



// and a /readme endpoint which will be open for the world to see

app.get('/readme', (req, res) => {

   res.json({ "message" : "This is open to the world!" })

})
```

```
app.listen(port,

  () => console.log(`Simple Express app listening on port ${port}!`))
```

Now that we have these endpoints, let’s re-run our application to ensure we haven’t broken anything:

$ node index.js

```
Simple Express app listening on port 3000!
```

Let’s open up **http://localhost:3000/secret** in our browser window now that this is up and running. You will see the following response:

http://localhost:3000/secret

```
{

  "message": "THIS IS SUPER SECRET, DO NOT SHARE!"

}
```

And when we open up the **/readme** endpoint, we should see the following:

http://localhost:3000/secret

```
{

  "message": "This is open to the world!"

}
```

Perfect, in this step we have been able to add 2 additional endpoints to our application that we will be attempting to protect with JWT validation.

Generating a Valid JWT

In this step, we are going to create 1 additional endpoint for our application that will return a valid JWT token for us to test out the authentication in subsequent steps.

In order for us to generate and validate these JWTs, we’ll need to first import the **jsonwebtoken** node module:

```
$ npm install --save jsonwebtoken
```

Once we have this node module installed, we can then go ahead and create a custom **/jwt** endpoint which will create a JWT token for us which features an incredibly simple payload **{ "body": "stuff" }** and is signed using a private key that I generated for a deprecated project.

Create a new file within your project direct called **private.pem** and add the following key:

private.pem

```
-----BEGIN RSA PRIVATE KEY-----

MIIEpAIBAAKCAQEAyVTQ9QxfutaYXKBbYfZbH2vhIWoIPEjAFSbsy1PZoIcclUQR

hJ8t2m7v47M8eEyYd7EvXTNdoN6CDs0DoNC9KESATZV5SUVr7sk9pOcMrm0VryAd

h7hQbbHWqyKmOehCt1JdX7gV2i5XnRb5qYQSyoB8sGdfR4SQ9q1XPRIpBP8RYXCP

WPmwnmtzYjfs+VVMp+ByNWgM8Qvyc3Z13tHKHWfTokbEbJJE0xfFG6CVFZy+T7uq

0UxhkWh8tekaDbfrIYtgC8HRHfRPcqvsN/pkJM0DIU4UNIf2TeLARa+iATNiF5IX

DeAAkIqoQL0tgt6S+8HIEPGiQ3gpGE56DON06QIDAQABAoIBAAvIxyJQwxmwjeJ+

EFs/jD3elqLaDflZWMTkLmAIXGik/+tMvKnCl3B9pdTyHMv9z77RxC/0XbqYy4wK

O/ghv7CnscrYwOyk/5hOdyk7zOY4xFgnzRKwmySQkDwcHxasnZsVWxnLMJxAsigj

vCFL9b2cn6/DnTQWclW996k/ct8z5DzodH/O+JyQl2MvLgjtf1OupbNZcjt4kMRP
```

```
GS5em5banlycUtPE0NAcS0QXleHLj5BXOp0AyD+PWfzhWcctiK7p+vmwXv6OVtlu
DIiTTOnsjoMq7skfVzRpjmAesllXBb1evwsrVK7cjYefEsVZ5DzOVLNU5Iug3HCf
RnftUAECgYEA6uWhJ9la1VpiYgEai4F61Rxs/MAQzmWVl0NQhPrczLCdRXfhmeBP
w1a4VkCD7FlZq/UuL4bSnU2JINL/enmrfZfdLdQgjY1VgPNki6v4Da9sqi+D17Wb
w863h/X29cukttFgUQdPO66bwsitoNVDsBqyf2hlFWC+rRPovoFA9WkCgYEA22s2
njRgoV6mTTCgXoZMs8unJ2CTd+BuVP1wDVsHR0VnK7blVlIW9Hop0ZmFK3I41kw3
3LXT6RSmZXsdgAsm3VuWlDVFY6t+JUUGYOfZZVNgUX0J597hf2SwxfJ8cgNHuSEO
glxc/dzHXMgamf8+84RCK3NSGP8xmuI/Wm5JE4ECgYEAmFelVTradlTQSc99b8zh
5SUyahoGzFWF1zyJFDW+zeIdndhKMIoSMRYlJ4tgBAFO7v9snNZL8kk/DlLJ7pzK
ZAICKJ7THfrz4VX5d7xofDexug5m65eVFkETNtKHAJK6mPbiCKs87/AmhQWx1gV6
iNRHv+ns5RiBka6/3A3oG0ECgYEAl+qIO0rqaG++1ozHTArSCl4DUlkkYQhLe56p
OSYASRE9WF/eM0DM0eHPGGahdC42OfE1cCOYH7WDa5mtGB0ggHxMKjsj2tk+kpFS
1D9SHjx24JShCiAfonNVjQfRr6KjwwKnKAzI+Z8ljRCikmLN9A5rPegvPE1by++/
i132TIECgYAiDd+OSokfZYi8mqjIoMI946+Hlb7lisb7CXksM69A+oWJrfkCwOMq
O3tExLbxtd7KZuAg1xH2thimIeT6tuCbCSAFGzr4EWCe7iavVzevHr98ivPLYXte
dyWl3YlvqO+SUuieGvHISlekblMO/4tcUscmKmo+FgkMroBsePcdEA==
-----END RSA PRIVATE KEY-----
```

Next, open up your **index.js** and add the following:

index.js

```
const jwt = require('jsonwebtoken');

const fs = require('fs')



// ... our other endpoints



app.get('/jwt', (req, res) => {

   let privateKey = fs.readFileSync('./private.pem', 'utf8');

   let token = jwt.sign({ "body": "stuff" }, "MySuperSecretPassPhrase", { algorithm: 'HS256'});

   res.send(token);

})





// ... the rest of our index.js
```

When we open up **http://localhost:3000/jwt** in our browser after restarting our application, we should see a valid JSON Web Token returned to us:

http://localhost:3000/jwt

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJib2R5Ijoic3R1ZmYiLCJpYXQiOjE1NTg4MTUyMzl9.3
Z8G-kCEfr2Xch1FnaEAOeywcVs6aJZ3sbCARybWCgM
```

Awesome, we now have a valid JWT that we can use to test the authentication we are going to be building in the next part of this tutorial!

Authentication Middleware

Let’s have a look at the authentication function which will look at an incoming request and parse it for an **Authorization** header.

index.js

```
function isAuthenticated(req, res, next) {

  if (typeof req.headers.authorization !== "undefined") {

    // retrieve the authorization header and parse out the

    // JWT using the split function

    let token = req.headers.authorization.split(" ")[1];

    let privateKey = fs.readFileSync('./private.pem', 'utf8');

    // Here we validate that the JSON Web Token is valid and has been

    // created using the same private pass phrase

    jwt.verify(token, privateKey, { algorithm: "HS256" }, (err, user) => {



      // if there has been an error...

      if (err) {

        // shut them out!

        res.status(500).json({ error: "Not Authorized" });

        throw new Error("Not Authorized");

      }
```

```
            // if the JWT is valid, allow them to hit

            // the intended endpoint

            return next();

        });

    } else {

        // No authorization header exists on the incoming

        // request, return not authorized and throw a new error

        res.status(500).json({ error: "Not Authorized" });

        throw new Error("Not Authorized");

    }

}
```

Now that we have this **isAuthorized** function, we can update our **/secret** endpoint to use this function as middleware.

index.js

```
app.get('/secret', isAuthorized, (req, res) => {

    res.json({ "message" : "THIS IS SUPER SECRET, DO NOT SHARE!" })

})
```

If we restart our application and try hitting the **/secret** endpoint once again, we should see the following response:

http://localhost:3000/secret

```
{
  "error": "Not Authorized"

}
```

Perfect, this has worked exactly as expected and we aren't able to access our **/secret** endpoint without the appropriate **Authorization** header set. Let's attempt to hit this endpoint with a **curl** command, we'll add the **Authorization** header and set it to **jwt** along with a valid JWT retrieved from our **/jwt** endpoint.

```
curl -H "Authorization: jwt " http://localhost:3000/secret
```

```
{"message":"THIS IS SUPER SECRET, DO NOT SHARE!"}%
```

Awesome, with this valid JWT in our **Authorization** header, we've been able to access the **/secret** endpoint!

Conclusion

So, we have managed to create an **isAuthenticated** middleware function that we can wrap our express API Endpoints with. This **isAuthenticated** function parses incoming **HTTP** requests to check if they have the appropriate **Authorization** header set and then checks to see whether the value of this header is a valid JWT token that has been signed with a specific private key.