Class 17 Bearer Authorization with JSON Web Tokens

Objectives

- Students will be able to create a Bearer Authentication parser
- Students will learn to implement Access Controls in their HTTP server routes

Venue Analogy

- Imagine you're attending a show.
- When you arrive to the show you tell the door person the performer has put your name on a list.
- The door person looks at their list, and finds your name
- The door person looks at your ID to prove your name
- The door person lets you in

Later on you step outside, and then try to reenter the venue.

- The door person doesn't recognize you. (They never do)
- You tell them you're on the list.
- They check their list agian, find your name and let you in.

You leave and re-enter several more times. The door person keeps forgetting you and they must keep checking their list. Everyone's tired of checking the list.

Their must be a better way!

A Better Venue

Another day you come back to the venue and find out a new manager has upgraded their door policies. They have a new system in place which they've found to be quite efficient.

- You arrive to a show where a perfomer has put you on a list.
- You tell the door person your name.
- The door person finds your name on their list.
- You show them your id to prove you're you.

Here's where things change!

- This time you get a bracelet!
- The door person says you can show them your bracelet and they'll know you've already been authorized to come in the venue.

You go in and out of the venue all night. Instead of giving your name and checking their list you walk through the door much more quickly because you simply have to show the door person the bracelet you're wearing.

Venue Analogy Explained

- Having your name on a list is like having your user information stored in a database.
- Providing your name and checking the list is like `Basic` authorization where you provide your username and password on every request.

What we'll see in today's lesson is that it's possible for the server to give users a `Token` that they can use later to prove that they've already been approved from the list.

`JSON Web Tokens` are passed through HTTP headers. The token contains information about who a user is, what a user is allowed to do, and they contain a special signature that proves the token came from the server.

Bearer Authorization

Previously we looked at `Basic Authorization`. It sent a username and a password to the server inserted as a header. The header looked like this:

```
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l
```

- Bearer authorization still uses the `Authorization` header
- It replaces the word `Basic` with `Bearer`
- It sends a `<token>` which we'll see contains much more data than simply a user name and password

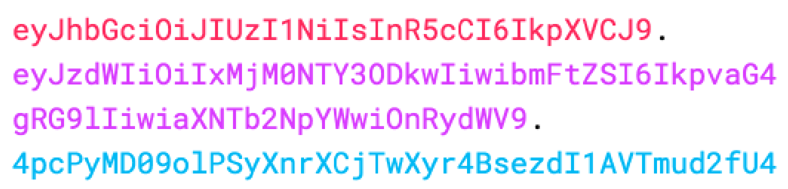
```
Authorization: Bearer <token>
```

JSON Web Token Structure

A JSON web token is a Base-64-encoded string. It contains three different sections of data, all concatenated together with a `.` dot.

There are `Header`, `Payload`, and `Signature` sections.

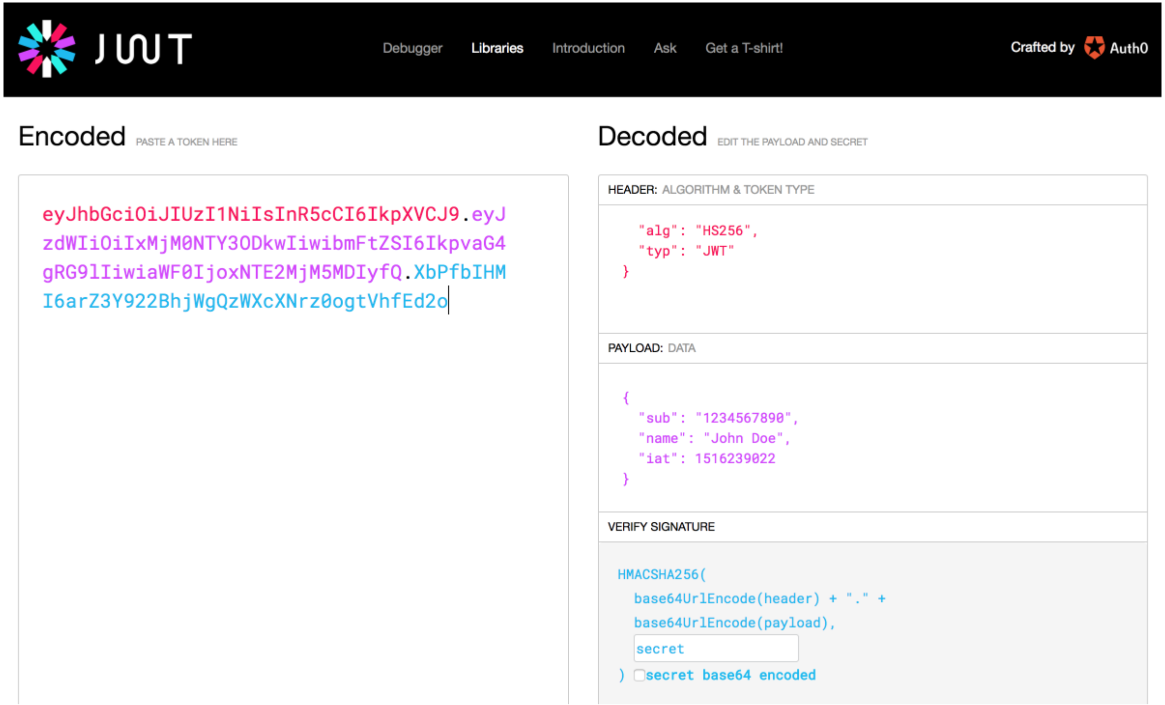
All together they might look this like:



```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Analyzing Token Structure

Go to jwt.io to use their excellent tool for analyzing the contents of a JSON Web Token.



Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.XbPfbIHM
I6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) □secret base64 encoded
```

The Header

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

The header is a small JSON object that contains information about how the JSON Web Token was created.

This header says that the algorithm `alg` used to encrypt the JSON Web Token is called `HS256`.

The Payload

```
{
  "exp": "Sunday, September 13, 2020 12:26:40 PM GMT",
  "username": "mayor@seattle.gov",
  "admin": true
}
```

The payload contains meta-data and information about the user.

This payload says three things:

- The entire token expires ("exp") at a certain time
- This token represents someone with the username
- The user with this token is an administrator

Strict formats of JSON Web Tokens define "claims" about users with all sorts of formal definitions. In this course we will simply not treat the payload so formally.

One example of a formal property in the payload would be the `"exp"` property which represents when a JSON webtoken will expire.

The Signature

- The signature helps verify the integrity of the data inside the header and the payload.
- The signature prevents users from rewriting their tokens
- For example, If the signature weren't present someone could change the `admin` property from `true` to `false`

`HMACSHA256` here represents one of many possible encryption algorithms.

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Code: How to Sign JWT Tokens

```
require('dotenv').config();  // import dotenv package for process.env access
const jwt = require('jwt');  // import the jwt package

let data = {username: 'mayor', isAdmin: true};
let token = jwt.sign(data, process.env.SECRET);
```

Code: How to Verify JWT Tokens

Create a piece of middleware you can use to verify the JSON Web Token and attach user information to the `req` request object so it's available in other routes.

lib/bearer-auth-middleware.js

```
function(req, res, next) {
  // read the user information from the "Authorization" header.
  var authHeader = req.headers.authorization;
  var token = authHeader.split('Bearer ')[1];

  jwt.verify(token, process.env.SECRET, (err, decoded) => {
    User.findOne({ username: decoded.username })
    .then(user => {
      req.user = user;

      // go to the next piece of middleware
      next();
    });
  });
}
```

The .env File

- We're beginning to have small pieces of configuration that exist in our code.
- It's nice to gather these all together and put them in once place.
- We can configure our servers to read configurations from a special `.env` file.
- `.env` files can contain confidential information so it's a good idea to add the `.env` to your `.gitignore`

We'll see ourselves add more sensitive information to `.env` files in the future when we begin deploying our servers publically.

(**NOTE:** properties on `process.env` can also be defined inside your bash environment, but we'll just be using the `.env` file.)

Run this if you want to stare into the sun and see all the properties attached to `.env`:

```
console.log(process.env)
```

.env Configuration and Access

```
PORT=3000
SECRET=moon
MONGODB_URI=mongodb://localhost/17-bearer-auth
```

Now you can access any of these properties by requiring and configuring the `dotenv` package and accessing these properties off a special `process.env` variable.

```
require('dotenv').config();  // import dotenv package for process.env access

console.log('PORT:', process.env.PORT);
console.log('SECRET:', process.env.SECRET);
console.log('MONGODB_URI:', process.env.MONGODB_URI);
```

Access Control

Now that we have users and secure authentication mechanisms on our servers we want to use them.

Think of a service like Twitter.

- People post tweets.
- People can read other people's tweets.
- People can delete their own tweets.
- People aren't allowed to delete other people's tweets.

`Access Control` defines what information users are allowed to access and manipulate.

We can use JSON Web Tokens to define what sort of data users are allowed to access and manipulate.

When someone requests to get all of a certain resource we can return only resources that they've created themselves, or are specifically allowed to access.

Check for Understanding

What's the name of the header used for both `Basic` and `Bearer` authorization?

Check for Understanding

What's the name of the header used for both `Basic` and `Bearer` authorization?

`Authorization`. The format for each specification looks like this:

```
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l
```

```
Authorization: Bearer <token>
```

Check for Understanding

What is the role of the Signature in a token?

Check for Understanding

What is the role of the Signature in a token?

The signature prevents users from tampering with the rest of the data in the JSON Web Token.