

Grabación

Presentación

Guardar los datos del lado del cliente

"Hoy, el doble factor de autenticación es fundamental en cualquier servicio que busque mejorar el nivel de autenticación de identidades. Solo usuario y contraseña no es suficiente para garantizar la identidad de un cliente, usuario, proveedor, etc. Con el agregado de otro factor de identidad (token digital, datos biométricos, token físico, etc.), el nivel de certeza de esa identidad crece exponencialmente." – [Luis Lubeck](#). *Especialista de seguridad informática del Laboratorio de Investigación de ESET Latinoamérica.*



Fuente

[Bryan Manuele](#), ingeniero Full Stack en Flexport, hace referencia en este [artículo](#) a una situación que era muy común en la autenticación: almacenar la información del usuario/a en cookies, una de las formas que se han utilizado por mucho tiempo. Pero para ello, el servidor debía almacenar esa información en una base de datos que consultaba cada vez que el/la usuario/a deseaba ingresar al sistema y verificar constantemente su acceso. Era a través de este proceso que se perdía escalabilidad y también seguridad en la aplicación ¡algo que queremos evitar! Para solucionar este problema, siguiendo a Bryan Manuele, surge otra forma que son los **JWT** que *"permiten almacenar de forma segura nuestros datos de sesión directamente en el cliente en forma de JWT."*

En la bitácora anterior nos adentramos en la seguridad de las APIs, distinguimos los conceptos de autenticación y autorización, y vimos cómo prevenir nuestros Endpoints de ataques comunes. ¡Siempre es recomendable encriptar los datos sensibles! En esta bitácora aprenderás un método específico para **codificar** y **decodificar** información: **JSON Web Token (JWT)**, un estándar abierto (RFC 7519) que define una forma más segura de transmitir información entre el cliente y el servidor.

Actualmente JSON Web Token (JWT) es una de las formas más utilizadas de autenticación. La principal ventaja de utilizar **token** es que se pueden generar de manera segura y pueden ser validados para confirmar su integridad sin que nadie ni nada los haya modificado. [Carlos Azaustre](#) –Ingeniero en Telemática y actualmente Senior Frontend Engineer en Eventbrite– menciona en su blog: *“ como los tokens son almacenados en el lado del cliente, no hay información de estado y la aplicación se vuelve totalmente escalable.”* Además, como indica su nombre, transmite esta información en formato JSON.

Veamos en siguiente ejemplo entre un cliente y el servidor:



El **JWT** se suele utilizar para la autorización de un cliente. El proceso funciona, por ejemplo, así:

1. El cliente envía su usuario y contraseña
2. El servidor lo autentifica y le retorna al cliente un texto. Este texto es una cadena de caracteres larga e ilegible, donde el servidor encripta un objeto JSON con una clave segura, y se lo envía.
3. Luego, en llamadas posteriores, el cliente envía este token y el servidor es capaz de decodificar ese token a los valores originales.

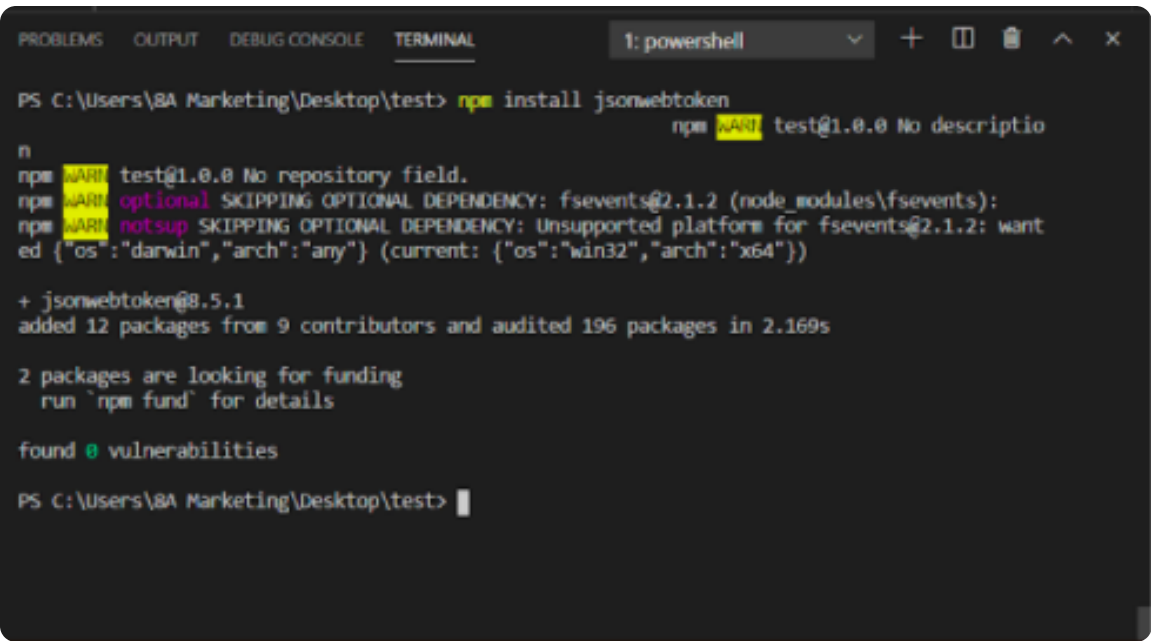
Conociendo JSON Web Tokens

El **JWT** está formado por tres partes de contenido: la **cabecera**, el **payload** o **contenido**, y la **firma**. Todos estos separados por un punto (.) :



En el contenido o cuerpo del mensaje es donde podemos declarar datos del usuario/a que queremos almacenar en formato JSON. Es útil para enviar el nombre del usuario/a, roles o accesos, por ejemplo. Hay que tener en cuenta que este contenido se almacena en Base64 y puede descodificar sin la necesidad de una clave, pero no puede ser verificado sin esa clave.

1. Para utilizar **JWT** en Node, debemos instalar el paquete [json web token](#) vía npm:



2. Para crear un **token**, primero debemos utilizar una firma o contraseña segura y posteriormente utilizar la función **sign** para generar el token con nuestros datos.

```
index.js - test - Visual Studio Code
index.js
1 const jwt = require('jsonwebtoken');
2 const informacion = {nombre: 'Manuel'}
3 const firma = 'Mi_password_secreto';
4 const token = jwt.sign(informacion, firma);
5 console.log(token);
6

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: node
PS C:\Users\BA Marketing\Desktop\test> nodemon .\index.js
[nodemon] 2.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .\index.js`
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21icmUiOiJNMWUw8iLCJpYXQiOiJlNjZgZmZlE3Mjd9.k2tpqIVLqkK_5QpxT9MMvZLByucAmelKyGahHkr8Vfig
[nodemon] clean exit - waiting for changes before restart
```


3. Para leer el contenido del token desde nuestro código, utilizamos la función `verify`.

```
index.js - test - Visual Studio Code
index.js
1 const jwt = require('jsonwebtoken');
2 const informacion = {nombre: 'Manuel'}
3 const firma = 'Mi_password_secreto';
4 const token = jwt.sign(informacion, firma);
5 const descodificado = jwt.verify(token, firma);
6 console.log(descodificado);
7


PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS C:\Users\BA Marketing\Desktop\test> nodemon .\index.js
[nodemon] 2.0.2
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .\index.js`
{ nombre: 'Manuel', iat: 1578671978 }
[nodemon] clean exit - waiting for changes before restart
```

Decodificar token

En su [página oficial](#), además de encontrar detalles de la implementación, lo más interesante es que disponen de una herramienta para codificar y decodificar un **token JWT**. En su servicio, para poder decodificar el token enviado por el/la usuario/a, validarlo con nuestra firma y obtener el contenido, utilizamos la función `verify`:



[Debugger](#) [Libraries](#) [Introduction](#) [Ask](#) [Get a T-shirt!](#)

Crafted by  Auth0

ALGORITHM

HS256

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJub21icmUiOiJlNjZgZmZlE3Mjd9.k2tpqIVLqkK_5QpxT9MMvZLByucAmelKyGahHkr8Vfig

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

{

"alg": "HS256",

"typ": "JWT"

}

PAYLOAD: DATA

{

"nombre": "Sergio",

"iat": 1564362836

}

VERIFY SIGNATURE

HMACHA256(
base64UrlEncode(header) + "." +
base64UrlEncode(payload),

AlgunSecret0

) ☐ secret base64 encoded

✔ Signature Verified

SHARE JWT

Implementar JWT en la API

Vamos a implementar un **método login** (post) en nuestra API que reciba un usuario/a y una contraseña en el body del request. En la función `validarUsuarioContraseña` verificamos que las credenciales sean correctas. En caso que lo sean, respondemos un JWT:

```
indexjs - test - Visual Studio Code
indexjs
newfile.txt
indexjs > app.post('/login') callback > error
1 app.post('/login', (req, res) => {
2   const { usuario, contrasena } = req.body
3   const validado = validarUsuarioContrasena(usuario, contrasena)
4
5   if (!validado) {
6     res.json({ error: 'Usuario o contraseña incorrecta' });
7     return;
8   }
9   const token = jwt.sign({
10    usuario
11  }, firmaSeguraJWT);
12
13  res.json({ token });
14 });
15
```

Una vez obtenido el token, vamos a agregarlo a los siguientes request para “avisarle” al servidor que estamos logueados en el header **Authorization** . En Postman se verá de la siguiente forma:

POST▼http://localhost:3000/seguro

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

▼ Headers (1)

	KEY	VALUE
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...

Creamos un método llamado **seguro en la API** y, a través de un **middleware**, verificamos el JWT del usuario/a:

```
indexjs - test - Visual Studio Code
indexjs
indexjs > ...
1 app.post('/seguro', autenticarUsuario, (req, res) => {
2   res.send('Esta es una página autenticada. Hola ${req.usuario.usuario}');
3 });
4
5 const autenticarUsuario = (req, res, next) => {
6   try {
7     const token = req.headers.authorization.split(' ')[1];
8     const verificarToken = jwt.verify(token, jwtSign);
9     if (verificarToken) {
10      req.usuario = verificarToken;
11      return next();
12    }
13   } catch (err) {
14     res.json({ error: 'Error al validar usuario' });
15   }
16 };
17
```

Si observas la imagen del Postman, verás que antepusimos la palabra Bearer al token que nos devolvió el servidor, la pregunta es ¿por qué?

El [RFC 6750](#) describe cómo utilizar un Token de autorización. Sigue el estándar, ¡mientras más estándares tengamos como developers más fácil nos será el mantenimiento para todos y todas!

¡Prepárate para el próximo encuentro!



Profundiza
Te invitamos a conocer más sobre el tema de esta bitácora.



Herramientas
Programas necesarios para facilitar tu experiencia.



Challenge

Te proponemos el siguiente desafío, ¿te animas?