Introducción a la autenticación por tokens

A la hora de proteger ciertas rutas o secciones de una aplicación web, podemos utilizar diversos mecanismos. En lo que a las aplicaciones web "tradicionales" se refiere (es decir, aquellas que sirven contenido visible en un navegador, como por ejemplo, y sobre todo, contenido HTML), el mecanismo de autenticación por antonomasia es la autenticación basada en sesiones.

Sin embargo, cuando queremos extender la aplicación web más allá del uso de un navegador, y permitir que otros tipos de clientes se conecten al *backend* (por ejemplo, aplicaciones de escritorio, o aplicaciones móviles), la autenticación basada en sesiones se queda corta o no sirve, y es necesario recurrir a otros mecanismos más universales, como la autenticación por tokens. Es el tipo de autenticación más utilizado en servicios REST.

1. Fundamentos de la autenticación por tokens

Un **token** por sí mismo es una cadena de texto que carece de significado. Pero, combinado con ciertas claves, es un mecanismo para proteger nuestras aplicaciones de accesos no permitidos. La autenticación basada en tokens es un método mediante el cual nos aseguramos de que cada petición a un servidor viene acompañada por un token firmado, que contiene los datos necesarios para verificar que el cliente ya ha sido validado previamente.

El mecanismo empleado para la autenticación por token es el siguiente:

1. El cliente envía sus datos de autenticación al servidor (típicamente un login y password).
2. El servidor valida esas credenciales contra algún tipo de almacenamiento (normalmente una base de datos). En el caso de que sean correctas, genera un token, una cadena codificada que permite identificar al usuario. Este token se envía al cliente, normalmente como datos de la respuesta. Internamente, puede contener algún dato que permita identificar al usuario, como su login o e-mail.

> **NOTA:** No conviene añadir en el token (ni tampoco en las sesiones) información muy confidencial, como el password, por ejemplo, ya que puede ser fácilmente descodificable. Esto no quiere decir que el token sea un mecanismo inseguro de autenticación, ya que el servidor utiliza una palabra secreta para cifrar una parte del token, y así poder verificar que es correcto, pero el resto del token sí queda más descubierto.

3. El cliente recibe el token y lo almacena de alguna forma localmente (mediante mecanismos como `localStorage` en JavaScript o similares en otros lenguajes). Ante cada nueva petición que se haga, el cliente reenvía dicho token en las cabeceras de la petición, para que el servidor verifique que es un cliente autorizado.

Normalmente a los tokens se les asigna un tiempo de vida o una fecha de caducidad (que pueden ser minutos, días, semanas... dependiendo de lo que nos interese y del tipo de aplicación). En cada nueva reconexión, el tiempo de vida se puede renovar (no es algo automático, deberemos hacerlo nosotros), y si

pasa más de ese tiempo estipulado sin que el cliente intente conectar, se le solicitará de nuevo que se autentifique.

Se suele emplear el estándar JWT (JSON Web Token), que define una forma compacta de transmitir esta información entre cliente y servidor empleando objetos JSON.

2. Estructura de un token JWT

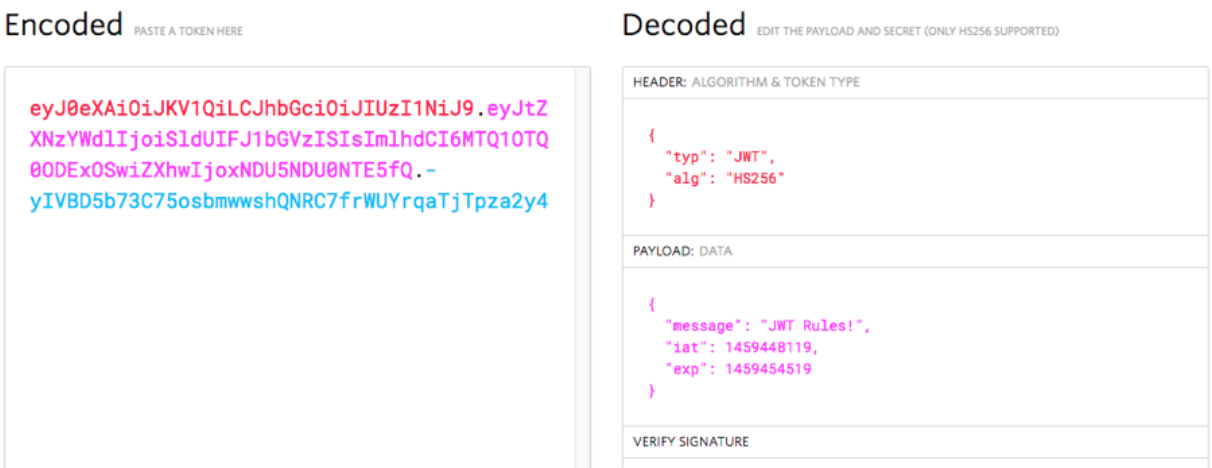
Un token JWT consiste en tres partes:

- **Cabecera (*header*)**: con información sobre el tipo de token generado (en este caso, JWT), algoritmo de encriptación (HS256, por ejemplo), etc.
- **Carga (*payload*)**: contiene la información codificada por el token (login, roles, permisos concretos, caducidad, etc), lo que se conoce como declaraciones (*claims*). Conviene no enviar información confidencial en este elemento, ya que, como veremos, es fácilmente descodificable.
- **Firma (*signature*)**: se emplea para validar el token y protegerlo frente a manipulaciones. Esta firma se genera mezclando tres elementos: la cabecera, el payload y una palabra secreta.

Se generará una cadena en formato *base64* con toda la información del token. Por ejemplo:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJtZXNzYWdlIjoiSldUIFJ1bGVzISIsImlhdCI6MTQ1
OTQ0ODExOSwiZXhwIjoxNDU5NDU0NTE5fQ.-yIVBD5b73C75osbmwwshQNRC7frWUYrqaTjTpza2y4
```

Si pegamos esta cadena en algún procesador de tokens, como el de jwt.io, podremos ver descodificadas la cabecera y el payload:



Esto quiere decir que no existe una encriptación para los datos que se envían (cabecera y payload), se pueden descodificar fácilmente desde *base64*. Pero sin conocer la palabra secreta que genera la firma, no se puede validar que ese token sea correcto. Esa palabra sólo la conoce el servidor. En el caso del ejemplo anterior, si utilizamos la palabra "L3@RNJWT" podremos obtener la validación del token en la página anterior.

Es decir, la tercera parte del token se emplea por el servidor para, una vez obtenidas la cabecera y el payload (descodificándolas desde *base64*) y uniendo a ellas la palabra secreta, comprobar que la cadena generada coincide con esa tercera parte. Si es así, el token es auténtico.

3. Ventajas e inconvenientes de la autenticación por token

El mecanismo de autenticación por token ofrece algunas ventajas respecto al tradicional método por sesiones:

- Los tokens no tienen estado (*stateless*), lo que significa que el servidor no debe almacenar en ninguna parte el registro de usuarios autenticados para comprobar si quien entra ha sido autorizado antes. La propia cadena que se envían cliente y servidor contiene toda la información. Por contra, la información de las sesiones (o los identificadores de cada sesión) sí se almacena en el servidor.
- Cualquier aplicación cliente soporta tokens, lo que supone que podemos emplear este mismo mecanismo para autenticar una aplicación de escritorio, móvil o web contra el mismo *backend*.
- Los tokens permiten especificar información adicional de acceso, como roles de usuario, permisos concretos a recursos, etc.

Sin embargo, para poder implementar una autenticación basada en tokens en una aplicación tradicional de navegador, necesitamos que el cliente no sea un cliente "tonto", que se limite a renderizar el contenido HTML y poco más. Es necesario cierto pre-procesamiento JavaScript en la parte cliente para enviar el token al servidor en cada petición, así como para recoger el token generado por el servidor cuando nos autenticamos. Esto hace imprescindible el uso de ciertas librerías en el cliente, como jQuery, Angular, React o Vue, entre otras.