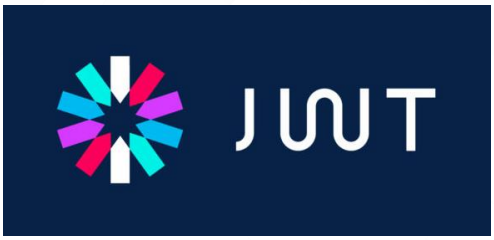

tokens JWT

JWT (JSON Web Token) es un estándar abierto (publicado en el RFC 7519) que define un método compacto y autocontenido para encapsular y compartir aserciones (claims) sobre una entidad (subject) de manera segura entre distintas partes mediante el uso de objetos JSON. Se puede confiar y verificar el contenido del token cuando este está firmado digitalmente (JWS, RFC 7515). La firma se puede generar usando claves simétricas (HMAC) o claves asimétricas (RSA o ECDSA). Adicionalmente los JWT pueden contener también datos cifrados (JWE, RFC 7516) para proteger datos sensibles, aunque este tipo de tokens no son objeto de este estudio.

Es importante resaltar que, por defecto, los tokens no están cifrados, y que la cadena que vemos es simplemente una serialización usando codificación base64url, que puede decodificarse fácilmente para ver el contenido JSON del token en claro.

La respuesta entonces a la pregunta inicial es 'depende...'. Como ocurre con otras muchas tecnologías, JWT depende fuertemente de la configuración que se use en la generación y del buen uso y la correcta validación de los tokens en el consumo.

JWT (JSON Web Token) es un estándar abierto que define un método compacto y autocontenido para encapsular y compartir aserciones sobre una entidad de manera segura entre distintas partes mediante el uso de objetos JSON.

Tipos de tokens y casos de uso

Comenzaremos viendo cuáles son los principales tipos de tokens y los principales casos de uso:

- Data token: En su forma serializada un JWT es muy compacto y fácil de transmitir en peticiones HTTP y se usa para el intercambio de datos.

- ID token: Emitido por un gestor de identidades, a petición de una aplicación cliente, tras haber autenticado a un usuario. Permite a la aplicación cliente obtener datos del usuario de manera confiable sin tener que gestionar sus credenciales.
- Access token: Emitido por un servidor de autorización, a petición de una aplicación cliente, y permite a esta el acceso a un recurso protegido en nombre de un usuario. Este token se usa como método de autenticación y autorización por parte de la aplicación cliente frente al servidor que aloja el recurso.

JWT permite el intercambio de datos de manera segura entre partes de manera más eficiente que otros estándares (SAML) debido a su menor tamaño, lo que lo hace ideal para los siguientes casos de uso:

- Intercambio de datos de sesión entre cliente y servidor: Debido a su mecanismo de serialización a veces son usados para transmitir información de sesión y estado entre el servidor y sus clientes. Se suelen usar " tokens inseguros" (sin firma).
- Autenticación federada: Elimina la necesidad de que las aplicaciones gestionen las credenciales de sus usuarios, delegando en un gestor de identidad de confianza el proceso de autenticación de los mismos. El gestor genera un token verificable por la aplicación que contiene los datos necesarios del usuario.
- Autorización de acceso: El token contiene la información necesaria para que un servicio de APIs pueda evaluar si la operación solicitada por el tenedor del token se puede permitir.

Cada caso de uso tiene distinto destinatario (aplicación cliente y servicio de API), pero en el caso de que se ejerza control simultáneo sobre las dos, un único token puede ser usado para ambos casos.

A continuación, vamos a enumerar las mejores prácticas cuando trabajamos con JWT, centrándonos sólo en la generación y la validación de los mismos.

Generación de token

Emitir siempre tokens firmados

Salvo muy contadas excepciones (para uso en el lado cliente para llevar información de sesión y datos para reconstruir interfaz de usuario) un token no debe emitirse sin firma. La firma es una protección básica que permite que los consumidores del token puedan confiar en él y asegurar que no ha sido manipulado.

Usar algoritmos criptográficos fuertes

A la hora de elegir el algoritmo de firma hay que tener en cuenta que los algoritmos de clave simétrica son vulnerables a ataques de fuerza bruta si la clave usada no es los suficientemente fuerte (nombres de mascotas y fechas de nacimiento tampoco valen para esto;-), con lo que hay que proporcionar suficiente complejidad si se eligen algoritmos de clave simétrica. Por otro lado los algoritmos de clave asimétrica simplifican la custodia de la clave, ya que esta sólo es necesaria en la parte servidora que genera el token.

Poner fecha de expiración e identificador único

Un token, una vez firmado, es válido para siempre si no hay una fecha de expiración (claim exp). En el caso de los "Access tokens", si alguien captura uno podrá tener acceso a la operativa permitida para siempre. El asignar identificadores (claim jti) a los tokens permite su revocación, en caso de compromiso del token es muy deseable tener la opción de poder revocarlo.

Dar valor a los emisores (issuer) y destinatarios (audience)

De cara a facilitar la gestión por parte de los consumidores de los tokens es necesario identificar el emisor (claim *iss*) y todos los posibles destinatarios (claim *aud*), de esta manera podrán identificar la clave de validación y comprobar que está dirigido a ellos. Como veremos más adelante, es una buena práctica que los consumidores del token validen estos datos.

No incluir datos sensibles sin cifrar en los claims

Como comentamos al inicio los tokens no van cifrados por defecto, así que hay que tener cuidado con la información que se introduce dentro de ellos. Si fuera necesario incluir información sensible o secretos hay que usar tokens cifrados.

A modo de ejemplo aquí os muestro la ejecución de un test para comprobar algunas de las validaciones que hemos visto usando jwt-checker:

```
$ docker run --rm bbvalabs/apicheck-curl http://my-company.com/auth/ | \
docker run --rm -i bbvalabs/jwt-checker -allowAlg HS256 -allowAlg HS384 \
-issuer bbva-iam -audience my-api-id -expiresAt 20200520T20:15:00\
-secret bXlTZWNyZXRQYXNzd29yZG15U2VjcmV0UGFzc3dvcmQK
```

En este ejemplo vemos una de las capacidades de la herramienta APICheck, que es el encadenamiento de acciones. En este caso la primera realiza una petición para obtener un token, generando un objeto de intercambio que le pasa a la siguiente herramienta, en este caso nuestro validador, que comprueba varios de los aspectos antes mencionados.

Validación del token

No aceptar tokens sin firma

La firma es el único medio de verificar que los datos contenidos dentro son de confianza. La primera validación que debemos hacer, después de verificar el formato, es que el token está firmado. Esta opción tiene que estar siempre activada, si no un atacante podría capturar un token, quitar la firma, modificarlo a su antojo y reenviarlo. No aceptar nunca un token con '*alg: "none"*' en su cabecera. La mejor protección es validar siempre que el campo *alg* contiene un valor de entre un conjunto específico que definamos, cuanto más pequeño mejor.

Validar claims de cabecera

Nunca debemos confiar en la inocuidad de la información recibida en la cabecera o en los claims, sobre todo si los vamos a usar para búsquedas en backends. Siempre debemos sanitizarlos antes de usarlos para evitar ataques de inyección, por ejemplo el valor del campo *kid* (identificador de clave) puede ser usado para la búsqueda en una base de datos (inyeccion SQL) y el valor de los campos *jku* (URL a la definición de una clave) y *x5u* (URL a la cadena de certificados de la clave) son URLs arbitrarias que pueden provocar ataques SSRF y que deberíamos validar contra una whitelist de URLs.

Validar siempre emisor y destinatarios

Antes de aceptar un token debemos verificar que está dirigido a nosotros (claim *aud*) y que ha sido emitido por la entidad esperada (claim *iss*), de esta manera reduciremos el riesgo de que un atacante use un token emitido para otro propósito pero cuya firma podamos verificar.

Almacenar las claves de firma por emisor y algoritmo

Cuando seleccionemos la clave para validar la firma debemos tener en cuenta no sólo el issuer si no también el algoritmo. Un atacante podría capturar un token que usa "*RS512*“ y modificarlo a “*HS256*", usando para firmar la clave pública del emisor (que es accesible). Si nosotros no hiciéramos esta validación podríamos aceptar como válido el token cuando realmente no lo es.