HINTS

TOKENS Y 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 ésta 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, siendo más eficiente que otros estándares, como SAML, esto es debido a un menor tamaño, haciéndolo 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, el cual 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 token se puede permitir.

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

IMPLEMENTANDO JWT CON LAS MEJORES PRÁCTICAS CENTRADAS EN LA GENERACIÓN Y LA VALIDACIÓN DE LOS MISMOS.

GENERACIÓN DE TOKEN

- **EMITIR SIEMPRE TOKENS FIRMADOS**

  Salvo contadas excepciones (uso en el lado cliente para llevar información de sesión y datos para reconstruir interfaz de usuario), un token no debe emitirse sin firma; pues ésta 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 es que la clave usada no es lo suficientemente fuerte (nombres de mascotas y fechas de nacimiento tampoco valen para esto), por lo que es necesario proporcionar suficiente complejidad si se eligen algoritmos de clave simétrica. Por otro lado, los algoritmos de clave asimétrica simplifican la custodia, ya que ésta sólo es necesaria en la parte servidora que genera el token.

- **COLOCAR 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 indefinido a la operativa permitida. El asignar identificadores (claim jti) a los tokens permite su revocación, y en caso de su compromiso, es muy deseable tener la opción de poder revocar.

- **DAR VALOR A LOS EMISOR (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, se podrá establecer la clave de validación, y comprobar que está dirigido a ellos. Como se revisará más adelante, es una buena práctica que los consumidores del token validen estos datos.

- **NO INCLUIR DATOS SENSIBLES SIN CIFRAR EN LOS CLAIMS**

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

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, pues si no un atacante podría capturarlo, quitar la firma, modificarlo a su antojo, y reenviarlo. Es importante no aceptar nunca un token con 'alg: "none"' en su cabecera; y 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 se usarán para búsquedas en backends. Siempre debemos protegerlos 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 (inyección 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, sino 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.

ALGORITMOS CRIPTOGRÁFICOS EN JWT QUE COMBINAN UN CIFRADO MÁS UN HASH, QUE ES MÁS SEGURO QUE USAR SOLO UN HASH:

- `HS256:` es el algoritmo usado por defecto, que consiste en un cifrado de clave simétrica HMAC (necesita una clave o SECRET para realizar el cifrado), con el algoritmo de hash SHA-256 (que produce una salida de 256 bits).
- `HS512:` cifrado de clave simétrica HMAC (necesita una clave o SECRET para realizar el cifrado), con el algoritmo de hash SHA-512 (que produce una salida de 512 bits).
- `RS256:` cifrado de clave simétrica RSASSA-PKCS1-v1_5, con el algoritmo de hash SHA-256.

  Los cifrados asimétricos son interesantes en aplicaciones desacopladas, donde podemos tener la clave privada en el servidor, y la clave pública en el cliente, por ejemplo.

- `RS512:` cifrado de clave asimétrica RSASSA-PKCS1-v1_5, con el algoritmo de hash SHA-512.
- `PS256:` cifrado de clave asimétrica RSASSA-PSS, con el algoritmo de hash SHA-256. Este es el reemplazo de RSA-PKCS.
- `PS512:` cifrado de clave asimétrica RSASSA-PSS, con el algoritmo de hash SHA-512.
- `ES256:` cifrado de clave asimétrica ECDSA, con el algoritmo de hash SHA-256. ECDSA utiliza claves más pequeñas, y es más eficiente que RSA. Actualmente, es el algoritmo que usan las criptomonedas Bitcoin y Ethereum, por ejemplo.
- `ES512:` cifrado de clave asimétrica ECDSA, con el algoritmo de hash SHA-512.