API Rest Security

Autenticación básica

- Esta es la forma más sencilla de asegurar tu API. Se basa principalmente en un nombre de usuario y una contraseña para identificarte.
- Para comunicar estas credenciales desde el cliente hasta el servidor, se debe realizar mediante el encabezado HTTP Autorización (Authorization), según la especificación del protocolo HTTP.
- Este método de autentificación puede ser un problema de seguridad en tu API REST.
- **Vulnerabilidad**
- Cualquiera que intercepte la trasmisión de datos puede decodificar fácilmente esta información. Esto se denomina **ataque Man-In-The-Middle (MiTM)**.
- Para proteger tu API mediante la autentificación básica debes configurar que las conexiones entre los clientes y tu servicio API funcionen únicamente mediante una conexión TLS/HTTPS, nunca sobre HTTP.

Ejemplo Basic

Autentificación basada clave API (API Key)

- A diferencia de los métodos de autenticación básica y token , en este caso primero debes configurar el acceso a los recursos de tu API. Tu sistema API debe generar una clave (*key*) y un *secret key* para cada cliente que requiera acceso a tus servicios. Cada vez que una aplicación necesite consumir los datos de tu API, deberás enviar tanto la *key* como la *secret key*.
- Este sistema es más seguro que los métodos anteriores, pero la generación de credenciales debe ser manual y esto dificulta la escalabilidad de tu API. La automatización de generación e intercambio de *key's* es una de las razones principales por las que se desarrolló el método de autentificación OAuth, que en el siguiente punto evaluaremos.

API Key

- Otros problemas con la autentificación basada en clave API es la administración de claves. Con tareas tan relevantes como:
- **a**. Genera la *key* y el *secret key*.
- **b**. Enviar las credenciales a los desarrolladores.
- **c**. Guardar de forma segura la *key* y el *secret key*.
- Puede ser complicado poder almacenar y administrar estas credenciales. Es por ello que es imprescindible contar con una **API Gateway**.

Autentificación basada en token

- En este método, el usuario se identifica al igual que con la autenticación básica, con sus credenciales, nombre de usuario y contraseña. Pero en este caso, con la primera petición de autentificación, el servidor generará un token basado en esas credenciales.
- El servidor guarda en base de datos este registro y lo devuelve al usuario para que a partir de ese momento no envíe más credenciales de inicio de sesión en cada petición HTTP. En lugar de las credenciales, simplemente se debe enviar el token codificado en cada petición HTTP.
- Por norma general, los tokens están codificados con la fecha y la hora para que en caso de que alguien intercepte el token con un ataque MiTM, no pueda utilizarlo pasado un tiempo establecido. Además de que el token se puede configurar para que caduque después de un tiempo definido, por lo que los usuarios deberán iniciar sesión de nuevo.

JWT - JSON Web Tokens

- JSON Web Token es un estandar abierto (RFC 7519) basado en JSON para crear tokens de acceso que permiten el uso de recursos de una aplicación o API. Este token llevará incorporada la información del usuario que necesita el servidor para identificarlo, así como información adicional que pueda serle útil (roles, permisos, etc.).
- Es posible usarlo para transferir cualquier datos entre servicios de nuestra aplicación y asegurarnos de que sean siempre válido. Por ejemplo si tenemos un servicio de envío de email otro servicio podría enviar una petición con un **JWT** junto al contenido del mail o cualquier otro dato necesario y que estemos seguros que esos datos no fueron alterados de ninguna forma.

FQ1
FQ2

Diapositiva 7

FQ1 Agregar
Fabricio Q; 10/6/2021

FQ2 JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.
Fabricio Q; 10/6/2021

¿Cuando deberíamos utilizar JWT?

- **Autorización**: este es el escenario más común para usar JWT. Una vez que el usuario haya iniciado sesión, cada solicitud posterior incluirá el JWT, lo que le permitirá acceder a rutas, servicios y recursos que están permitidos con ese token. El inicio de sesión único es una función que utiliza ampliamente JWT en la actualidad, debido a su pequeña sobrecarga y su capacidad para usarse fácilmente en diferentes dominios.
- **Intercambio de información**: los JWT son una buena forma de transmitir información de forma segura entre las partes. Debido a que los JWT se pueden firmar, por ejemplo, utilizando pares de claves públicas / privadas, puede estar seguro de que los remitentes son quienes dicen ser. Además, como la firma se calcula utilizando el encabezado y la carga útil, también puede verificar que el contenido no haya sido manipulado.

Ventajas de JWT sobre Cookies

JSON Web Token ofrece varias ventajas en comparación con el método tradicional de autentificación y autorización con *cookies*, por lo que se utiliza en las siguientes situaciones:

- **Aplicaciones REST**: En las aplicaciones REST, el JWT garantiza la ausencia de estado enviando los datos de autentificación directamente con la petición.
- **Intercambio de recursos de origen cruzado**: JSON Web Token envía información mediante el llamado *cross-origin resource sharing*, lo cual le da una gran ventaja sobre las *cookies*, que no suelen enviarse con este procedimiento.
- **Uso de varios *frameworks***: JSON Web Token está estandarizado y puede utilizarse una y otra vez. Cuando se emplean múltiples *frameworks*, los datos de autentificación pueden compartirse más fácilmente.

Estructura de un JWT

- Los JWT tienen una estructura definida y estándar basada en tres partes:

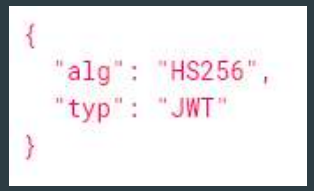
header.payload.signature

- Las primeras dos partes (header y payload) son strings en base64 creados a partir dos JSON. La tercer parte (signature) toma las otras dos partes y las encripta usando un algoritmo (normalmente SHA-256). Ejemplo: password ‘secret’

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

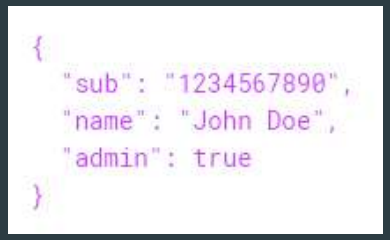
JWT Header

- El header de un JWT tiene la siguiente forma:



- La propiedad alg indica el algoritmo usado para la firma y la propiedad typ define el tipo de token, en nuestro caso JWT.
- Ejemplos de algoritmos de firma HMAC SHA256 or RSA.
- Propiedades:
- Tipo de token (typ) - Identifica el tipo de token.
- Tipo de contenido (cty) - Identifica el tipo de contenido (siempre debe ser JWT)
- Algoritmo de firmado (alg) - Indica que tipo de algoritmo fue usado para firmar el token.

JWT Payload



- La información se proporciona como pares *key/value* (*clave-valor*); las claves se denominan ***claims*** en JWT. Hay tres tipos diferentes de *claims*:
- Los ***claims*** **registrados** son los que figuran en el IANA JSON Web Token Claim Register y cuyo propósito se establece en un estándar. Se utilizan nombres de *claim* cortos para abreviar el *token* lo máximo posible. No son mandatorios
- Los ***claims*** **públicos** pueden definirse a voluntad, ya que no están sujetos a restricciones. Para que no se produzcan conflictos en la semántica de las claves, es necesario registrar los *claims* públicamente en el JSON Web Token Claim Register de la IANA o asignarles nombres que no puedan coincidir.
- Los ***claims*** **privados** están destinados a los datos que intercambiamos especialmente con nuestras propias aplicaciones. Si bien los *claims* públicos contienen información como *nombre* o *correo electrónico*, los ***claims*** **privados son más concretos**. Por ejemplo, suelen incluir datos como *identificación de usuario* o *nombre de departamento*. Al nombrarlos, es importante asegurarse de que no vayan a entrar en conflicto con ningún *claim* registrado o público.

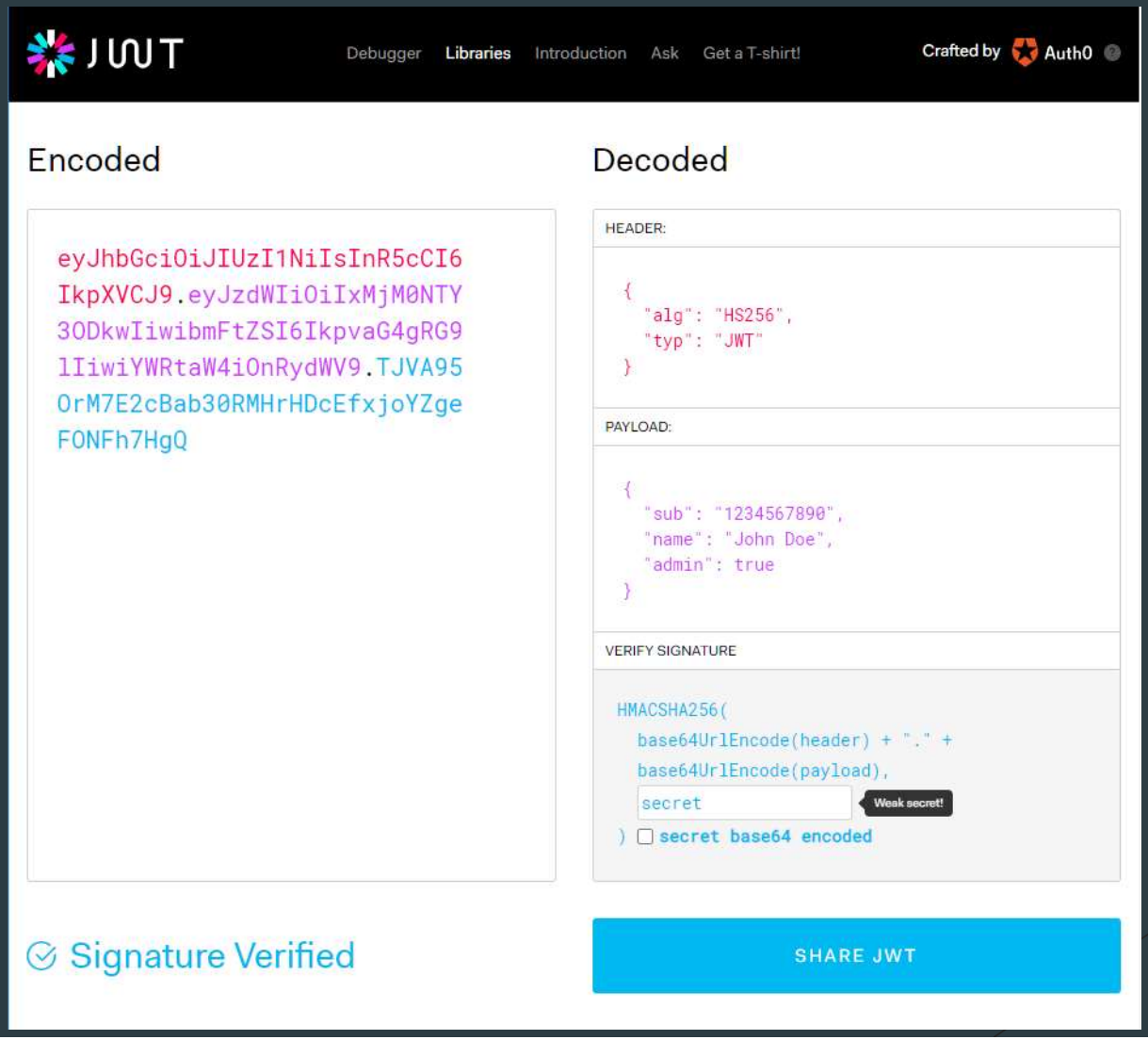
JWT Payload

- **Propiedades estándar o Claims Registrados**
- Creador (iss) - Identifica a quien creo el JWT
- Razón (sub) - Identifica la razón del JWT, se puede usar para limitar su uso a ciertos casos.
- Audiencia (aud) - Identifica quien se supone que va a recibir el JWT. Un ejemplo puede ser web, android o ios. Quien use un JWT con este campo debe además de usar el JWT enviar el valor definido en esta propiedad de alguna otra forma.
- Tiempo de expiración (exp) - Una fecha que sirva para verificar si el JWT esta vencido y obligar al usuario a volver a autenticarse.
- No antes (nbf) - Indica desde que momento se va a empezar a aceptar un JWT.
- Creado (iat) - Indica cuando fue creado el JWT.
- ID (jti) - Un identifador único para cada JWT.

JWT Signature

- La firma del **JWT** se genera usando los anteriores dos campos en base64 y una key secreta (que solo se sepa en los servidores que creen o usen el **JWT**) para usar un algoritmo de encriptación.
- De esta forma obtenemos la firma y la agregamos al final de nuestro **JWT**.

JWT Debugger (https://jwt.io/)



Oauth 2.0

- OAuth2 es un protocolo de autorización que permite a terceros (clientes) acceder a contenidos propiedad de un usuario (alojados en aplicaciones de confianza, servidor de recursos) sin que éstos tengan que manejar ni conocer las credenciales del usuario. Es decir, aplicaciones de terceros pueden acceder a contenidos propiedad del usuario, pero estas aplicaciones no conocen las credenciales de autenticación .
- OAuth 2.0 es un método de autorización utilizado por compañías como Google, Facebook, Twitter, Amazon, Microsoft, etc. Su propósito es permitir a otros proveedores, servicios o aplicaciones, el acceso a la información sin facilitar directamente las credenciales de los usuarios. Pero tranquilo, únicamente accederán bajo la confirmación del usuario, validando la información a la que se le autorizara acceder.

OAuth 2.0

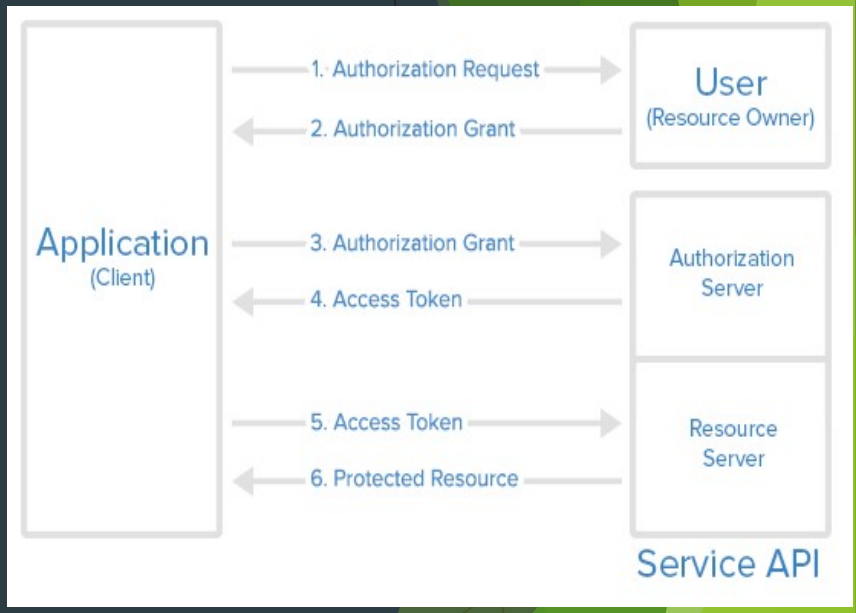
Dentro de una autenticación OAuth existen los siguientes actores:

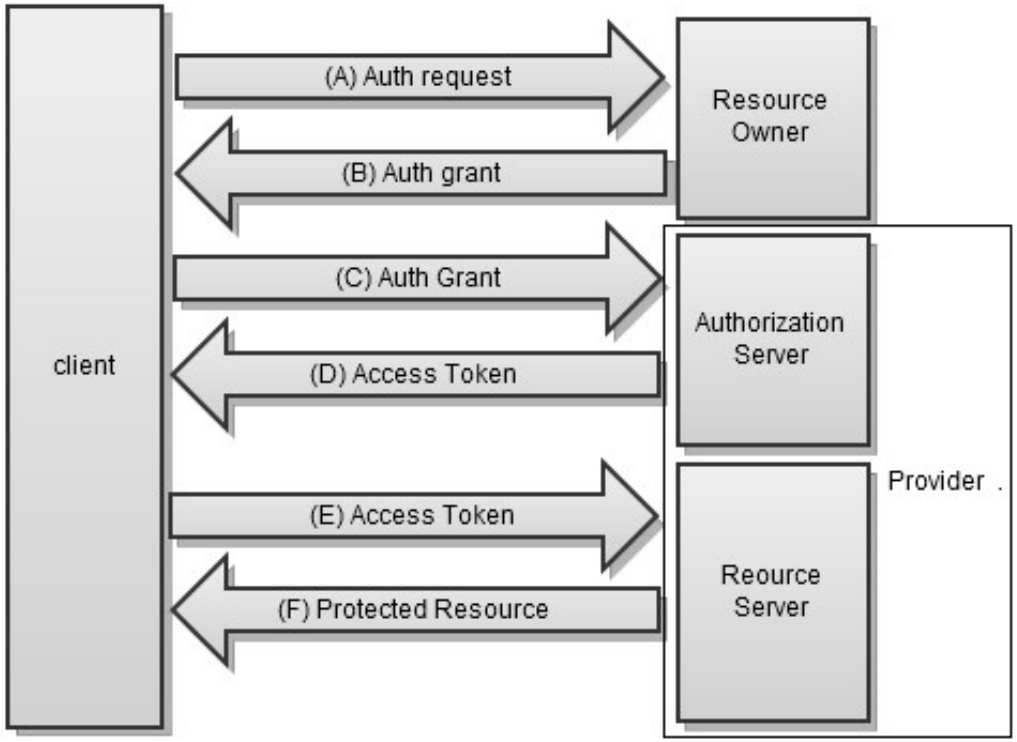
•**Resource Owner:** Propietario de recursos. Es el usuario (o una aplicación en su nombre) que cuenta con la propiedad o el acceso a los recursos que quiere acceder. Cuando es una persona nos referiremos a él como usuario final.

•**Client**: Aplicación cliente, es la encargada de realizar las peticiones en nombre del resource owner, el cliente será el encargado de realizar todas las peticiones necesarias para comprobar que efectivamente el que está haciendo uso de él es el resource owner.

•**Authorization Server**: Servidor de autorización. Es el encargado de verificar las credenciales del resource owner y en caso de ser correctas emitir los tokens que representan a los usuarios.

•**Resource Server**: Servidor de recursos. Es el servicio o servidor que contiene los recursos protegidos, debe de poder comunicarse con el servidor de autorización para poder validar si la petición de acceso a sus recursos es valida o no.





(A) El cliente solicita autorización al propietario del recurso. La petición se puede realizar directamente al servidor de autorización en lugar de al propietario del recurso, lo que es aconsejable.
(B) El propietario del recurso concede la autorización para acceder al recurso informando al cliente uno de los cuatro tipos de autorización disponibles (authorization code, implicit, resource owner password credentials or client credentials). El tipo de concesión depende del tipo de concesión pedido por el cliente al iniciar el flujo
(C) El cliente pide al servidor de autorización un token de acceso, identificandose y presentando la autorización obtenida en el paso B.
(D) El servidor de autorización valida las credenciales del cliente y la autorización. Si son válidas devuelve un token de acceso.
(E) - (F) El cliente y el servidor de recursos ya son capaces de intercambiar peticiones seguras con el token de acceso para servir contenido protegido.

¿Por qué OAuth2?

- OAuth2 tiene dos grandes ventajas, soluciona el problema de la confianza entre un usuario y aplicaciones de terceros, y a su vez permite a un proveedor de servicios/API facilitar a aplicaciones de terceros a que amplien sus servicios con aplicaciones que hacen uso de los datos de sus usuarios de manera segura y dejando al usuario la decisión de cuando y a quien, revocar o facilitar acceso a sus datos, creando así un ecosistema de aplicaciones alrededor del proveedor de servicios/API.

  La confianza, o la falta de ella, es el motivo principal para querer implementar un protocolo de autorización como OAuth2. La falta de confianza de un usuario con una aplicación de terceros puede propiciar que el usuario quiera permitir a la aplicación realizar tareas y obtener datos en su nombre pero sin darle las credenciales de autenticación a dicha aplicación

EndPoints en el proveedor

- El proveedor puede definir dos puntos finales para permitir todo el flujo de autorización-validación (en algunos casos los dos son necesarios y en otros únicamente el token-endpoint es necesario). Uno de ellos será el encargado de autentificación y validación de credenciales y el otro será el encargado de expedir tokens de acceso.
- **Authorization end-point**: Este punto de acceso debe ser el encargado de validar clientes y propietarios de recursos. Cuando un cliente quiere acceder a un recurso, es en este punto en el que el proveedor valida tanto el cliente como el propietario y genera concesiones para acceder a los recursos protegidos.
- **Token end-point**: este punto es el encargado de dar tokens de acceso, normalmente a cambio de concesiones previamente expedidas por el Authorization end-point.

Obteniendo permisos para acceder al recurso protegido

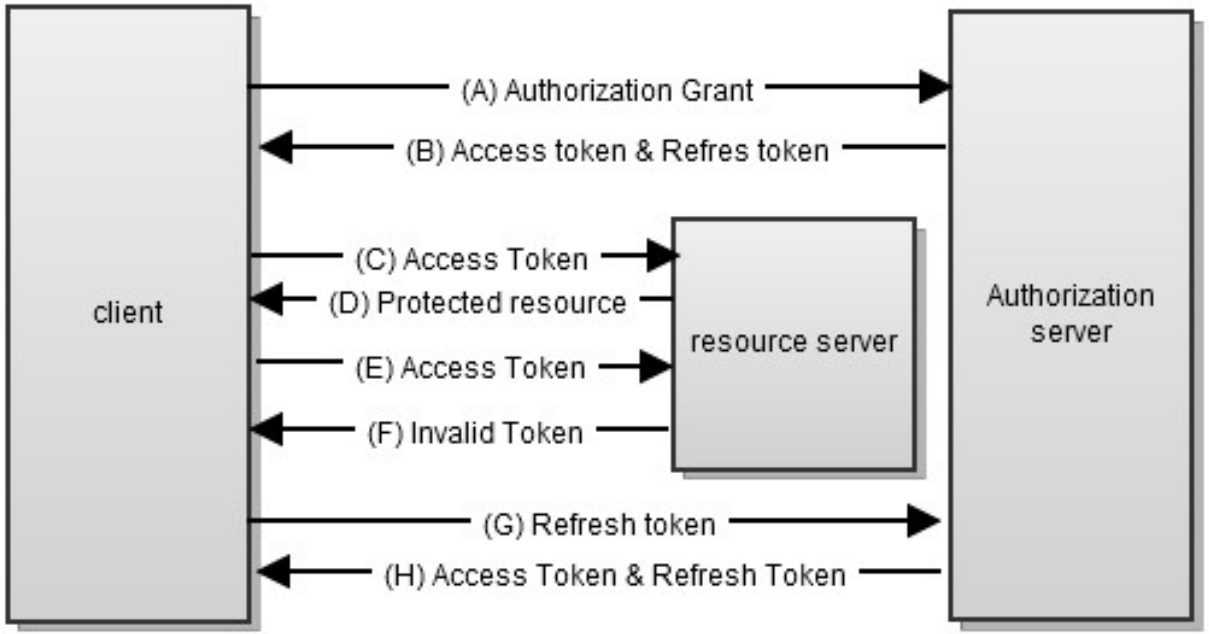
- Lo primero que necesita es solicitar permisos al propietario del recurso. Este paso se denomina **Authorization Grant**. Para que un servidor de recursos te conceda un token de acceso, es imprescindible presentarle un Authorization Grant junto a las credenciales del cliente que intenta acceder a dichos datos. Un proveedor de servicios no tiene que implementar obligatoriamente todos los tipos de concesiones (Grant Type). Cada uno de los tipos de concesiones está pensado para resolver las necesidades de los distintos tipos de clientes, dependiendo del escenario donde nos encontremos.
- **Ámbito del permiso**
- OAuth2 define un parámetro que puede intervenir en la negociación de autorización llamado ámbito (scope). Es el proveedor el encargado de definir los distintos scopes (o tomar la decisión de no usarlos) y la manera de obtener el permiso para acceder a cada uno de ellos. Mediante los scopes se consigue un mecanismo para delimitar el rango del acceso concedido al cliente, pudiendo darle acceso solo a determinados recursos del sistema.

Tipos de concesión

- **Authorization Code:** Se utiliza cuando **aplicaciones web** intentan acceder al servidor de recursos. (Server to Server). Tenemos que tener presente que el Resource Owner siempre se autentifica y valida en el authorization server y las credenciales nunca se exponen al cliente. Una vez el propietario de los recursos ha sido autentificado y validado en el proveedor, mediante la uri de redirección se devuelve al cliente un código de acceso que puede intercambiar por un accessToken.
- Implicit: Éste método se utiliza sobre todo en aplicaciones cliente (browser-based-application) para reducir el número de llamadas. Éste **Authorization Grant** es el menos seguro de todos, ya que se evita la doble comprobación.
- **Credenciales del propietario del recurso:** se utiliza cuando el cliente es total confianza y el usuario confía en él tanto como para darle sus credenciales de acceso al Servidor de Recursos. Éstas credenciales no se almacenan en la parte cliente, únicamente se utiliza para obtener un token de acceso.
- **Credenciales del cliente:** En este escenario únicamente interviene el cliente y el servidor de autorización (2-legged). Se utiliza cuando es el mismo cliente el que quiere acceder a datos del servidor de autorización sin necesidad de hacerlo en nombre de un propietario de recursos.

Obteniendo un nuevo token de acceso

- Los token de acceso deben tener un periodo de expiración después del cual se consideran caducados y el proveedor debe recharzarlos, obligando al cliente a obtener un nuevo token de acceso. Para evitar tener que volver realizar todo el proceso de validación necesario para la obtención del token, entra el mecanismo de **Refresh Token.** Cuando el proveedor concede un token de acceso, puede incluir un token de refresco asociado al token de acceso. Mediante este token de refresco se puede obtener un nuevo token de acceso válido.
- Con el mecanismo de refresh token, conseguimos que el Resource Owner solo deba autorizar al cliente una única vez, pudiendo generar nuevos token de acceso a partir del refresh token y la validación del cliente al que previamente ya se le habían dado permisos por parte del propietario del recurso.



(A) El cliente pide un token de acceso al proveedor, presentando un Authorization Grant

(B) El Authorization server valida la petición y devuelve un access token

(C) El cliente pide recursos al servidor de recursos presentando el acces token.

(D) El servidor de recursos valida el access token y si es válido devuelve los recursos protegidos

(E) Los pasos C y D se repiten hasta que el servidor de recursos detecta un token de acceso expirado o inválido. Cuando el cliente recive una respuesta de token de acceso inválido, salta al paso G

(F) Cuando el token de acceso es inválido el servidor de recursos se lo informa al cliente

(G) El cliente hace una petición para obtener un nuevo token de acceso presentando el refresh token

(H) el servidor de autorización autentica al cliente y valida el token de refresco. Si todo es válido, devuelve un acces token con un token de refresco opcional.

Ejemplo JWT